人工智能基础教程 Python篇



丁亮、姜春茂、于振中◎编著

清华大学出版社

人工智能基础教程：Python 篇（青少版）

丁　亮　姜春茂　于振中　编著

清華大學出版社
北　京

内 容 简 介

《人工智能基础教程：Python 篇（青少版）》全书分为 2 篇——人工智能编程基础篇、人工智能篇。人工智能编程基础篇包括：初识 Python、基本数据类型、Python 的流程控制、数组操作、文件操作、绘制需要的图表、函数、面向对象、异常、集合与概率、学点统计学、数据管理与分析；人工智能篇包括人工智能导论、初识机器学习、自然语言处理、语音识别技术、计算机视觉、人工神经网络。

本教程是以人工智能为主线，融合学科特点进行编程能力的培养。读者可以通过本教程结合我们开发的在线编程平台完成课程内容和辅助内容的学习。也可在后期以我们自主开发的机器人为载体，进行进一步驱动人工智能的实验。

本书每章都配备了相关练习，帮助读者巩固所学习的知识。

本书不仅讲解了人工智能基础，还包括了入门 Python 编程的必要知识。可以作为高中生课程教材，也可作为青少年自学人工智能基础和 Python 编程基础的参考书。

图书在版编目（CIP）数据

人工智能基础教程. Python 篇：青少版/丁亮，姜春茂，于振中编著. —北京：清华大学出版社，2019
ISBN 978-7-302-51649-1

Ⅰ. ①人…　Ⅱ. ①丁…　②姜…　③于…　Ⅲ. ①人工智能-青少年读物　②软件工具-程序设计-青少年读物　Ⅳ. ①TP18-49　②TP311.561-49

中国版本图书馆 CIP 数据核字（2018）第 258162 号

责任编辑：贾小红
封面设计：杜广芳
版式设计：楠竹文化
责任校对：马军令
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×230mm　　印　　张：21.5　　字　　数：427 千字
版　　次：2019 年 2 月第 1 版　　印　　次：2019 年 2 月第 1 次印刷
定　　价：49.80 元

产品编号：082016-01

前　言

2017 年国务院印发了《关于印发新一代人工智能发展规划的通知》，提出要围绕教育、医疗、养老等迫切民生需求，加快人工智能创新应用。要发展智能教育，利用智能技术加快推动人才培养模式、教学方法改革，构建包含智能学习、交互式学习的新型教育体系。更进一步提出要实施全民智能教育项目，在中小学阶段设置人工智能相关课程，逐步推广编程教育，鼓励社会力量参与寓教于乐的编程教学软件、游戏软件的开发和推广。在目前的国际国内大背景下，人工智能教育不仅是个人成长的需要，更是实现创新型国家发展的需要，是我国成为世界强国的需要，而人工智能教育必将从中学阶段展开。

人工智能教育的方式和方法，可以大致分为如下两种：一是在信息技术课程中开展部分人工智能基础知识的内容教学；二是在具备一定数学基础的高中阶段开展人工智能的选修课程。然而，单纯的知识性、科普性的人工智能知识的讲授缺少实践过程，缺少对动手能力的培养。因此，我们编写了本套教材。

本套教材以人工智能教育为主线，以 Python 编程为实现手段，以在线编程训练平台和人工智能案例实现平台为辅助工具，同时辅以我们自主开发的一系列机器人为载体，打造立体沉浸式的人工智能学习体系。之所以选择 Python 语言为实现手段，主要是考虑到 Python 具有与人工智能天然的切合性，而且语言本身入门相对容易，特别适合高中阶段的学生学习。

乐学系列机器人（中学系列）是由哈工大机器人（合肥）国际创新研究院独立开发的具有自主知识产权的机器人产品。该系列产品可以自由组装，自由编程。在锻炼动手能力的同时具有学科特色，能够促进物理、数学学科的学习；乐学系列机器人支持 Python 编程，具有丰富的传感器，可以实现人工智能的绝大部分要求，包括视觉、语音、自然语言处理、推理、逻辑、深度学习等系列的功能。学生可以在乐学系列机器人上围绕人工智能进行编程学习，通过机器人可以直观地观察智能编程的效果。总地来说，本套教材是以人工智能为主线，融合学科特点的编程能力培养；以自主开发的机器人为载体，驱动人工智能的实践；以信息素养的提升为内涵目标，以小组工程项目为牵引，实现高中生团队合作能力的提升。

本套教材目前规划总计两册，第一册为《人工智能基础教程：Python 篇（青少版）》，主要讲授 Python 的编程基础及其人工智能初步，在这部分内容中，我们结合高中教育的学

科特点和高中学生的认知特点，不求 Python 内容的大而全，但求够用和编程思想的训练。在学生具备了 Python 的基础以后，讲授了人工智能的基础知识，包括自然语言处理、语音识别、计算机视觉、人工神经网络等内容。与此同时，我们还开发了在线学习和竞赛平台，学生可以通过教材和在线编程平台完成课程内容和辅助内容的学习。随后将出版的第二册为《人工智能实践教程：机器人篇（青少版）》，主要讲解人工智能在我们自主开发的机器人上的实现，通过在机器人上的实现，让学生真正懂得如何去开发一个个人工智能的小案例，同时我们为学生提供了扩展性的开发案例来扩展他们的思维，唯有思维能力的提高才能促进创造力的提升。综合案例需通过团队的配合和合作完成，将极大地促进学生的学习和能力的提高。

本册教材共 18 章，分为 2 篇，第 1 篇为人工智能编程基础篇，选择 Python 作为入门人工智能的基本语言，在这一部分中，我们紧密结合人工智能和高中教育的学科特点，力争融知识、趣味、能力培养为一体。结合高中教育的学科特点，学生在学习的过程中就可以解决数学等学科中出现的问题和难点，有利于提高学生的学科学习和科学素养。第 2 篇为人工智能篇，该部分中我们对人工智能涉及的诸多要素如机器学习算法、自然语言处理、语音识别、视觉识别、神经网络等内容进行讲解，讲解的方式是通过 Python 的实践进行讲授，每个要素和内容的呈现过程包括基础知识、代码实现等；限于篇幅，我们不可能把所有的人工智能的内容都呈现给大家，但是期待这些基础内容可以帮助大家打下基础，以便未来进一步扩展学习。

致教师：在编写教材之初，我们深知对于高中的信息技术课教师而言，学习一门新的编程语言并将它传授给学生是一件成本较高的事情，但是时代在进步，社会在发展，培养学生的创新能力已经成为我国经济社会发展的重要一环。国家层面的政策必将导向到考核学生的思维、创新方向上来，已经有诸多发达省份的中高考开始考查 Python 编程的内容，在全国铺开只是时间问题而已。因此，我们期待每位老师抓住机会，和我们一起迎接挑战。

致学生：Python 是目前通用的编程语言中相对简单易学的，而且支持它的第三方功能库特别丰富，项目开发速度非常快，所以广泛应用于各个领域。一旦学会了，你的思考和运用方式会让人刮目相看！期待同学们在快乐中学到真正的编程知识，还能把编程发展为特长，在以后的工作和学习中都能成为你最最实在的加分项。

教学建议：首先，如果把本书作为高中的编程基础教材，可以学习第 1 篇的内容，即第 1～12 章，这部分内容从 Python 的基础讲起，融合高中的数学、物理、化学等学科知识，结合在线的学习平台，学生可以获得很好的编程素养训练。其次，学有余力的学校和学生，

可以安排人工智能部分的学习和机器人的实践案例，真正体会到人工智能带给我们的直观体验，在一个立体的学习环境中达到科学、技术、工程等知识的融合，对于学生的未来发展具有诸多好处，对于建设特色驱动的学校也是重要的切入点。

教学资源：本套教材包括《人工智能基础教程：Python 篇（青少版）》《人工智能实践教程：机器人篇（青少版）》。哈工智诚在线编程云平台不仅提供了在线编程、综合案例实践功能，还包括了视频课程讲解、虚拟仿真实验室等。

本套丛书的编写由哈工大机器人（合肥）国际创新研究院统一组织，本册教材由丁亮教授、姜春茂教授担任主编，曲明成博士、刘鹏飞副研究员和夏科睿副研究员担任副主编；其中丁亮教授编写了第 17、18 章，姜春茂教授编写了第 1～12 章，曲明成博士编写了第 13、14 章，刘鹏飞副研究员和夏科睿副研究员分别编写了 15、16 章。在此还要特别感谢于振中老师对本书编写的帮助和支持。在编写过程中，众多的研究生也付出了辛勤的劳动，他们是吴俊伟、栾浩、王凯旋、徐晓霞、张彤等，在此对他们表示感谢。

由于时间紧张，错误在所难免，期待各位读者提出宝贵意见和建议。

目　　录

第 1 篇　人工智能编程基础篇

第 2 篇 人工智能篇

第 1 篇



人工智能编程基础篇

人工智能（Artificial Intelligence，AI）是计算机科学的一个重要分支，它希望生产出一种类似于人类智能性质的智能机器。人工智能的研究范围很广，包括机器人、语言识别、图像识别、自然语言处理等众多领域。在日常生活中，人工智能已得到广泛应用，如智能音箱、无人驾驶汽车等。

Python 语言称得上是最适合人工智能开发的编程语言，主要原因是，它简单易用，可以无缝地与数据结构和众多的人工智能算法一起使用。

让我们一起在 Python 的世界里徜徉吧！

第 1 章　初识 Python

通过本章，我们将在学习 Python 的道路上迈出第一步。本章将要学习：

- Python 的历史以及优缺点。
- 完成 Python 的开发环境搭建。
- Python 的程序解释机制。
- 编写并运行第一个 Python 程序。
- Python 基础。

1.1　Python 的前世今生

Python 是一种面向对象的解释型计算机程序设计语言，由荷兰人 Guido van Rossum（吉多·范罗苏姆）于 1989 年发明，它的第一个发行版于 1991 年发布。

1989 年圣诞节期间，在阿姆斯特丹，Guido van Rossum 为了打发圣诞节的无聊决心要开发一种新的编程语言，因为他是 Monty Python 喜剧团体的爱好者，所以选用了 Python 作为该语言的名字。

就这样，Python 在 Guido van Rossum 手中诞生了。由于它的简洁性、易读性以及可扩展性，Python 已经成为最受欢迎的程序设计语言之一，一些知名大学早已采用 Python 来教授程序设计课程。众多开源的科学计算软件包都提供了 Python 的调用接口。

1991 年，第一个由 C 语言实现的 Python 编译器诞生。

1994 年，Python 1.0 版本发布。

2000 年，Python 2.0 版本发布，现有的 Python 语言框架已经完成。

2010 年，Python 2.7 版本发布，这是 Python 2.X 的最后一个版本。

2008 年 12 月 3 日，Python 3.0 版本发布，对比 Python 2.X 版本做出了较大改动。

2016 年 12 月 23 日，Python 3.6 版本发布。

2018 年 3 月，Python 宣布 Python 2.7 版本将于 2020 年 1 月 1 日终止支持。不过请放

心，本书使用的案例都是以 Python 3.X 版本编写的。

1.2　Python 的优势

随着计算机技术的不断发展，目前有众多可选的计算机语言。作为入门者，如何选择一门适合自己的编程语言是至关重要的。相对来说，Python 对于初学者具有以下几点优势：

（1）软件质量。Python 注重可读性、一致性和软件质量，这使得它的代码易于理解。同时，Python 支持软件开发的高级重用机制。

（2）提高开发效率。相对于很多编译类型语言，Python 的开发效率会提高很多，因为它录入更少的代码、调试更少的代码以及维护更少的代码。

（3）组件集成。Python 有着灵活的集成机制，这使得它可以调用 C 或者 C++的库或是被 C/C++调用，也可以和 Java 组件集成。Python 绝不是一个独立的工具。

（4）易于学习。Python 的易用性和强大的内置工具让编程变成了一种乐趣，在本书的学习中你将会体会到。

计算机语言

计算机语言（Computer Language）指用于人与计算机之间通信的语言，它可以分为机器语言、汇编语言、高级语言三大类。计算机语言是人与计算机之间传递信息的媒介。为了使计算机顺利地进行各种工作，需要有一套用以编写计算机程序的数字、字符和语法规则，由这些字符和语法规则组成计算机的各种指令（或各种语句）。这些就是计算机能接受的语言。

20 世纪 40 年代当计算机刚刚问世的时候，程序员必须手动控制计算机。当时的计算机十分昂贵，唯一想到利用程序设计语言来解决问题的人是德国工程师 Konrad Zuse（康拉德•楚泽）。不久后，计算机的价格大幅度下降，而计算机程序也越来越复杂。也就是说，开发时间已经远比运行时间来得宝贵。于是，新的集成、可视化的开发环境越来越流行。它们减少了所付出的时间、金钱（以及脑细胞）。只要轻敲几个键，一整段代码就可以使用了。这也得益于可以重用的程序代码库。随着 C、Pascal、Fortran 等结构化高级语言的诞生，使程序员可以离开机器层次，在更抽象的层次上表达意图。随着程序规模的不断扩

大，在 60 年代末期爆发了软件危机，在当时的程序设计模型中都无法克服错误随着代码的扩大而级数般地扩大，以至到了无法控制的地步，这个时候就出现了一种新的思考程序设计方式和程序设计模型——面向对象程序设计，由此也诞生了一批支持此技术的程序设计语言，如 Eiffel、C++、Java，这些语言都以新的观点去看待问题，即问题就是由各种不同属性的对象以及对象之间的消息传递构成。面向对象语言由此必须支持新的程序设计技术，如数据隐藏、数据抽象、用户定义类型、继承、多态等。

1.3　Python 的缺陷

前面介绍了很多 Python 的优点，在这里还是要说一下 Python 的缺陷。与 C/C++这类的编程语言相比，Python 的执行速度还不够快。

现如今，即使 CPU 的处理速度很快，在一些应用领域仍强调程序的执行速度。如果今后涉及这些领域的话，可以通过分离部分需要优化速度的应用，将其转换为编译好的扩展，并在系统中使用 Python 脚本将这些应用连接起来。当然，这些都是后话了。

1.4　Ubuntu 下开发环境的搭建

本节要讲的是在 Ubuntu 16.04 下搭建开发环境，在这里介绍两种方式：通过 Ubuntu 中自带的 apt-get 命令安装和通过 PyEnv 安装。

1.4.1　通过 apt-get 命令安装

这种方式非常简便，只需打开终端，输入如下命令，再按 Enter 键。

命令：

```
sudo apt-get update && sudo apt-get install python3
```

1.4.2　通过 PyEnv 安装

PyEnv 是一个简单的 Python 版本管理工具，它的前身是 Pythonbrew。通过 PyEnv 可以

改变全局的 Python 版本，安装、管理多种不同的 Python 版本。

PyEnv 的安装方式也很简便，一般有两种安装方法：一种是通过 Git 命令安装，另一种是直接下载安装。

注意：

这两种方法都是在终端中完成的。在进入 Ubuntu 系统后使用 Ctrl+Alt+T 快捷键可以快速打开终端。

1. 通过 Git 命令安装 PyEnv

命令：

```
1:  sudo apt-get install git
2:  git clone git://github.com/yyuu/pyenv.git .pyenv
3:  echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
4:  echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
5:  echo 'eval "$(pyenv init -)"' >> ~/.bashrc
6:  exec $SHELL –l
```

分析：

由于本书主要讲的并不是 shell 语句，这里将简单地介绍这个过程，有兴趣的读者可以查阅资料了解详情。Git 是一个开源的分布式版本控制系统，它用于敏捷高效地处理任何或小或大的项目。第 1 行语句就是将 Git 安装到 Ubuntu 系统中。第 2 行及其后续的语句是使用 Git 安装 PyEnv。

2. 直接下载安装 PyEnv

命令：

```
curl-L https://raw.githubusercontent.com/yyuu/pyenv-installer/master/bin/pyenv-installer | bash
```

分析：

这里使用 curl 命令从服务器上下载数据并运行安装。这种方法在使用时，过程耗时可能会比较长。

通过上述两种方法我们已经将 PyEnv 安装到系统中了，接下来，使用 pyenv install -list 命令来看看都可以安装哪些 Python 版本。命令详情如图 1.1 所示。

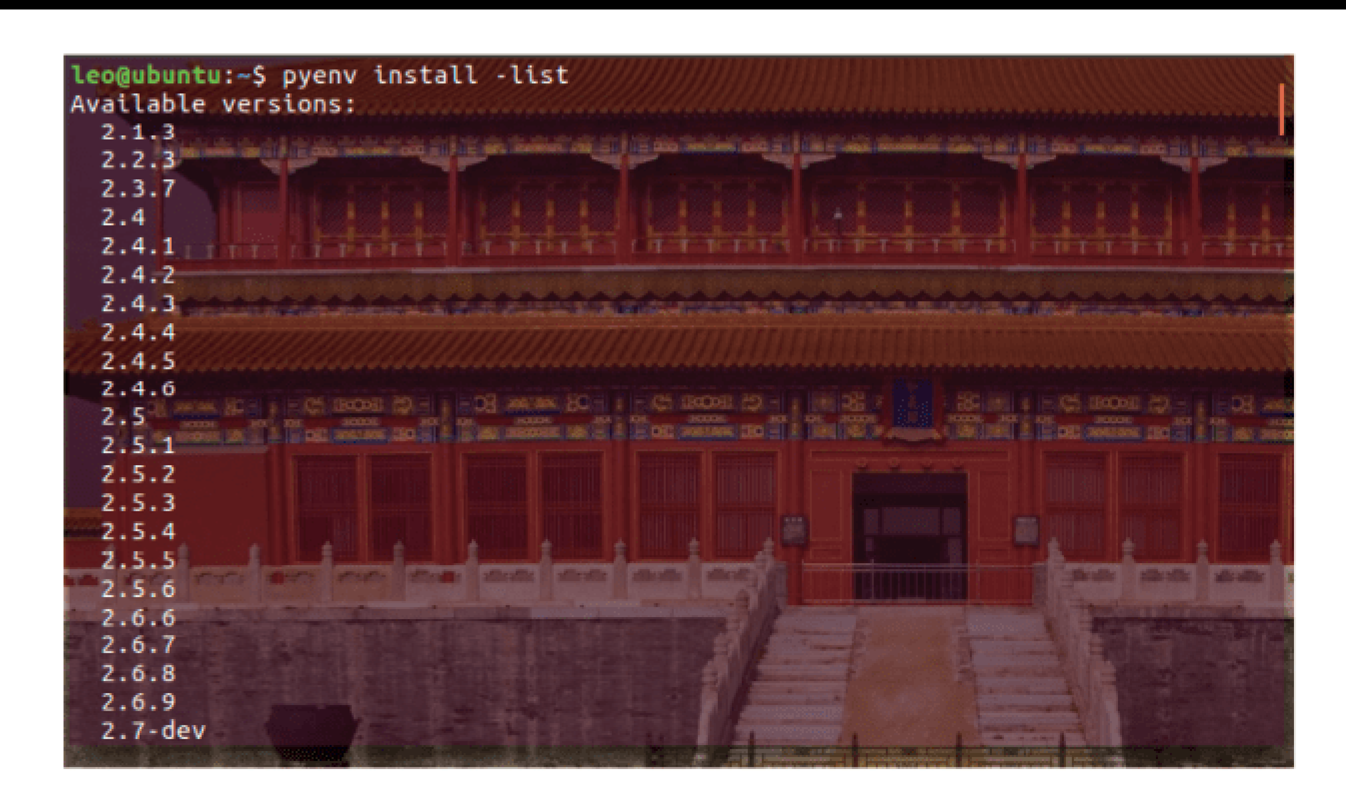


图 1.1　使用 pyenv 命令查看可安装版本（截图中仅是部分版本）

3. 安装 Python

我们选用 Python 3.6.4，但是在安装之前先要进行一些前序工作：安装依赖。

安装依赖：

```
1:    sudo apt-get install libc6-dev gcc
2:    sudo apt-get install -y make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-dev libsqlite3-dev
wget curl llvm
```

经过这个过程之后，就可以安装 Python 3.6.4 了。

安装 Python 3.6.4：

```
pyenv install 3.6.4 -v
```

至此，已经完成了安装过程（过程可能会有些缓慢）。可以使用环境刷新命令更新一下。

刷新：

```
1:    pyenv rehas
2:    pyenv version
```

接下来，就可以很便捷地使用 PyEnv 管理 Python 版本了。由于 Ubuntu 16.04 中自带了 Python 2.7.2，所以在使用时，需要切换一下。具体操作如图 1.2 所示。

```
leo@ubuntu:~$ python  --version
Python 2.7.12
leo@ubuntu:~$ pyenv global 3.6.4
leo@ubuntu:~$ python  --version
Python 3.6.4
```

图 1.2　使用 pyenv global 3.6.4 命令切换版本

完成 Python 的安装之后，再选择一款适合的编译器，可以让我们在编写程序时更得心应手。

虽然在 Ubuntu 中可以用 Vim 或者 Emacs，但是为了将学习的重心放在学习 Python 而不是编辑器上，本书推荐使用 PyCharm 编译器进行 Python 语言的学习，下面将介绍该编译器的安装方法。

1.4.3　安装 PyCharm

PyCharm 教育版是一款非常适合初学者学习 Python 的免费开发工具，下载网址为：https://www.jetbrains.com/pycharm-edu/。

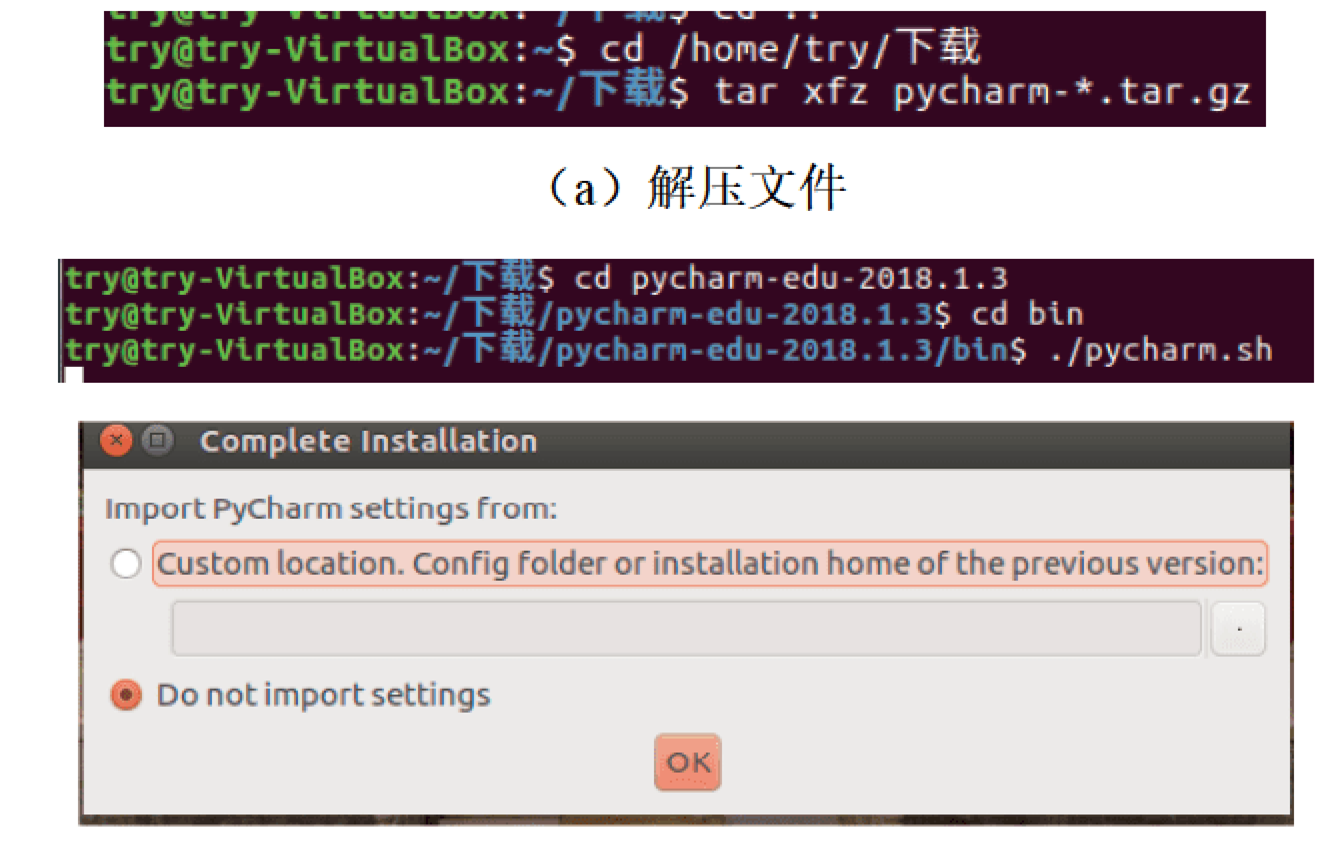
PyCharm 的安装如下。

进入下载网址页面，单击页面中央的 DOWNLOAD FREE 按钮下载 pycharm-edu-2018.1.3.tar.gz 压缩文件（或者是下载当前最新版本都可以）。

下载完成后，进入终端并使用 cd 命令切换到 pycharm 压缩包所在路径〔见图 1.3（a）〕，并用下面的命令解压。

```
tar xfz pycharm-*.tar.gz
```

解压完成之后，在终端继续使用 cd 命令进入到解压文件夹中的 bin 文件所在路径下，并运行./pycharm.sh 命令进行安装。这一过程演示如图 1.3（b）所示。



（a）解压文件

（b）安装 PyCharm

图 1.3　PyCharm 的安装过程

完成上述步骤之后，我们就进入了 PyCharm 的设定界面中，这里你可以根据自己的喜好设置。

完成设定后，我们来一起创建一个项目。启动 PyCharm，单击界面中的 Create New Project 超链接，接下来进入创建工程界面，在 Location 文本框中可以选择工程的路径。在 Location 下面的 Project Interpreter: New Virtualenv environment 前面有一个三角按钮，单击它，就可以在 Base interpreter 一栏中选择我们要使用的 Python 版本（见图 1.4）。

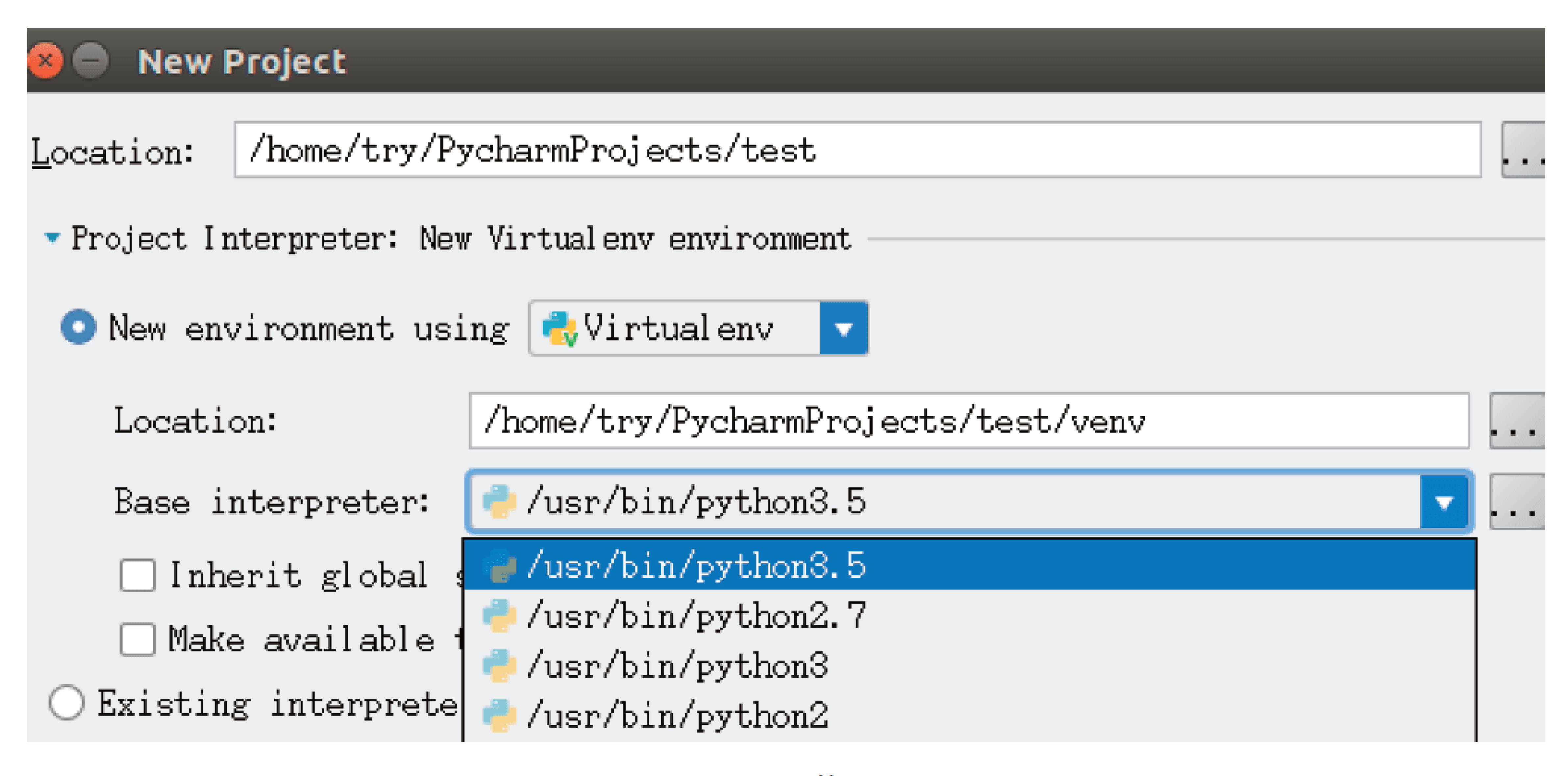
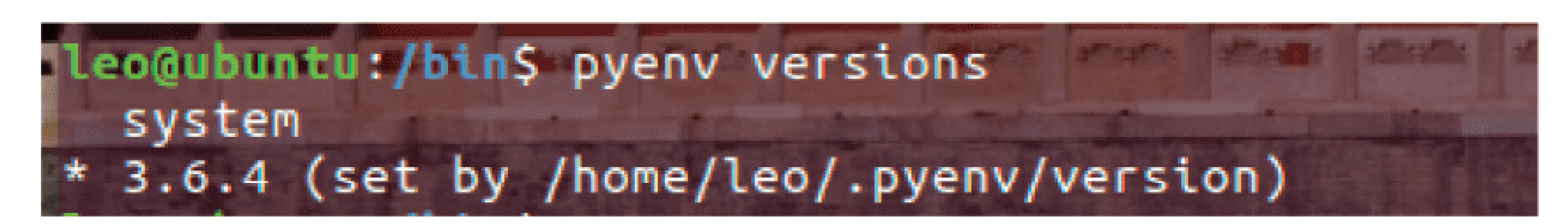
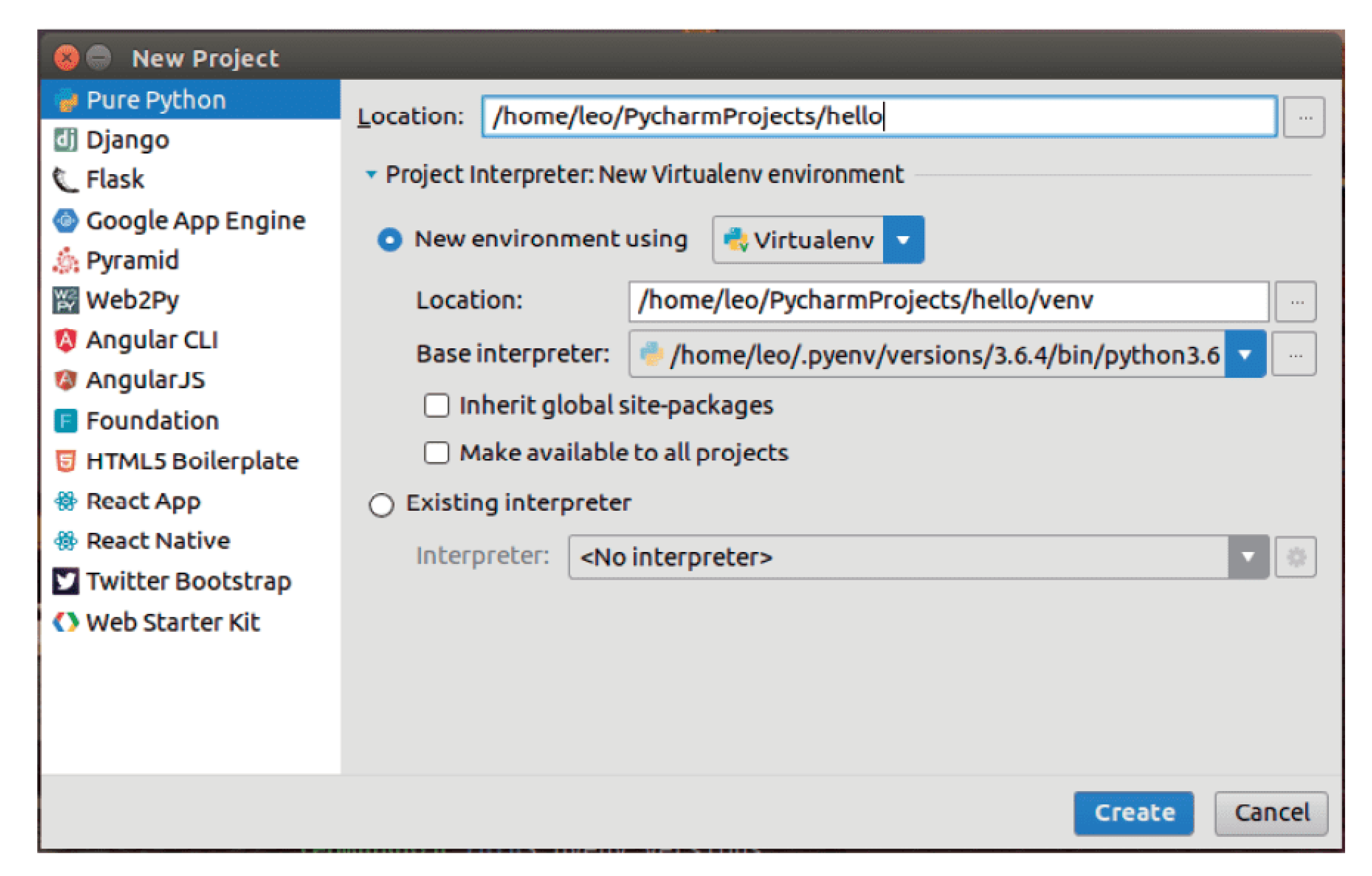


图 1.4　创建项目及其相关设置选择

如果使用的是 PyEnv 安装的 Python 版本，可以通过 pyenv versions 命令（见图 1.5（a））来查看都安装了哪些版本以及版本文件的路径。我们通过文件的路径来找到对应版本的路径（因为安装版本的路径和设置文件的路径都在 pyenv 文件夹下）。具体情况如图 1.5（b）所示。



（a）查看已经安装的版本



（b）PyCharm 中的工程设置

图 1.5　创建工程及其设置

1.5　Windows 下开发环境的搭建

本节将详细介绍 Windows 下开发环境的搭建。相比较上一节在 Ubuntu 下开发环境的搭建，Windows 下开发环境的搭建就十分简单了。

1.5.1　安装 Python

首先进入官网的下载页面，网址为 https://www.python.org/downloads/，并单击页面中的 Download Python 3.6.5 按钮下载安装程序（在写作本书时，最新版本是 3.6.5，以后若有更新可下载最新版。也可以在 Looking for a specific release?中找到 Python 3.6.5 版本进行下载），如图 1.6 所示。

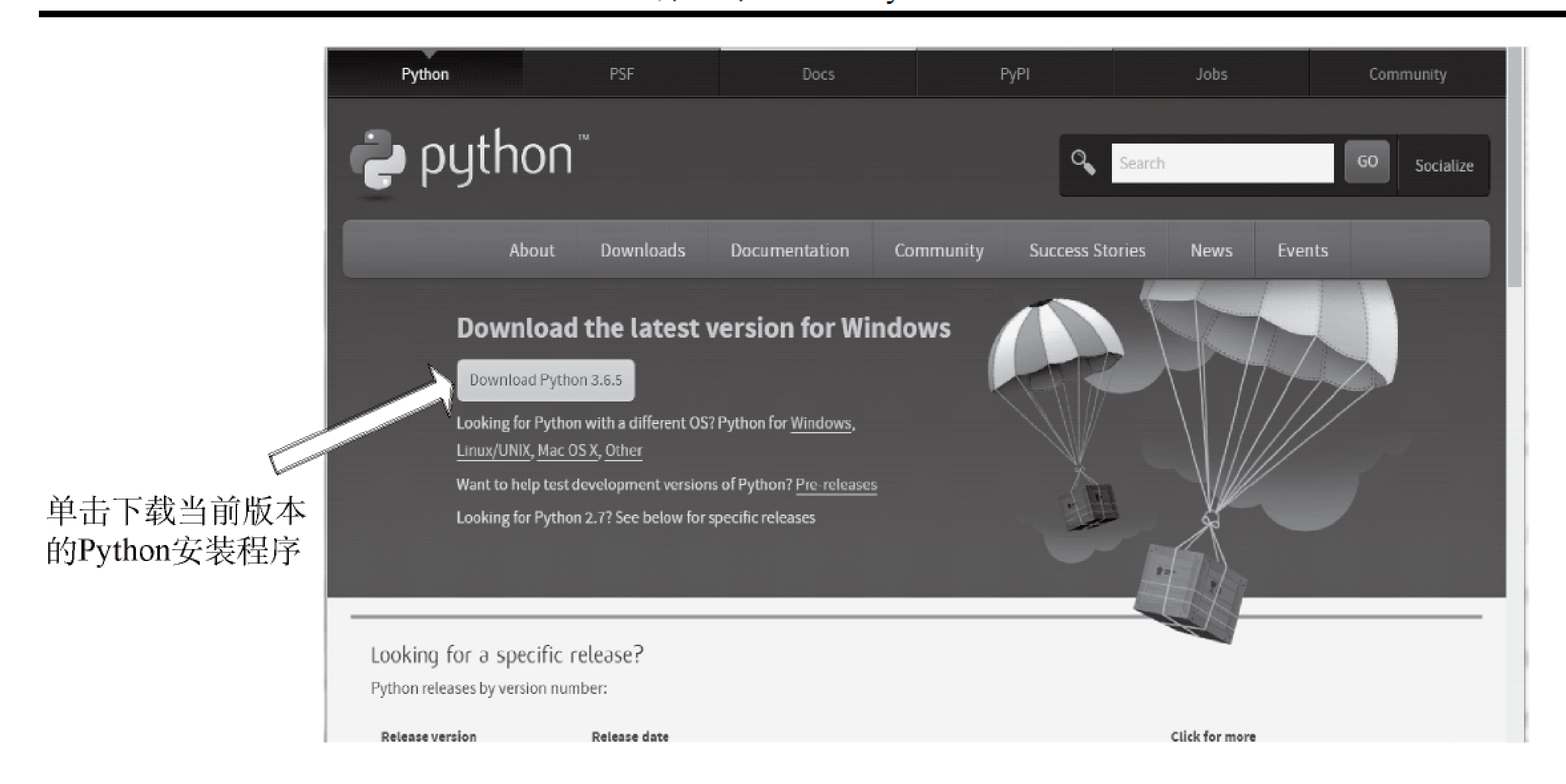
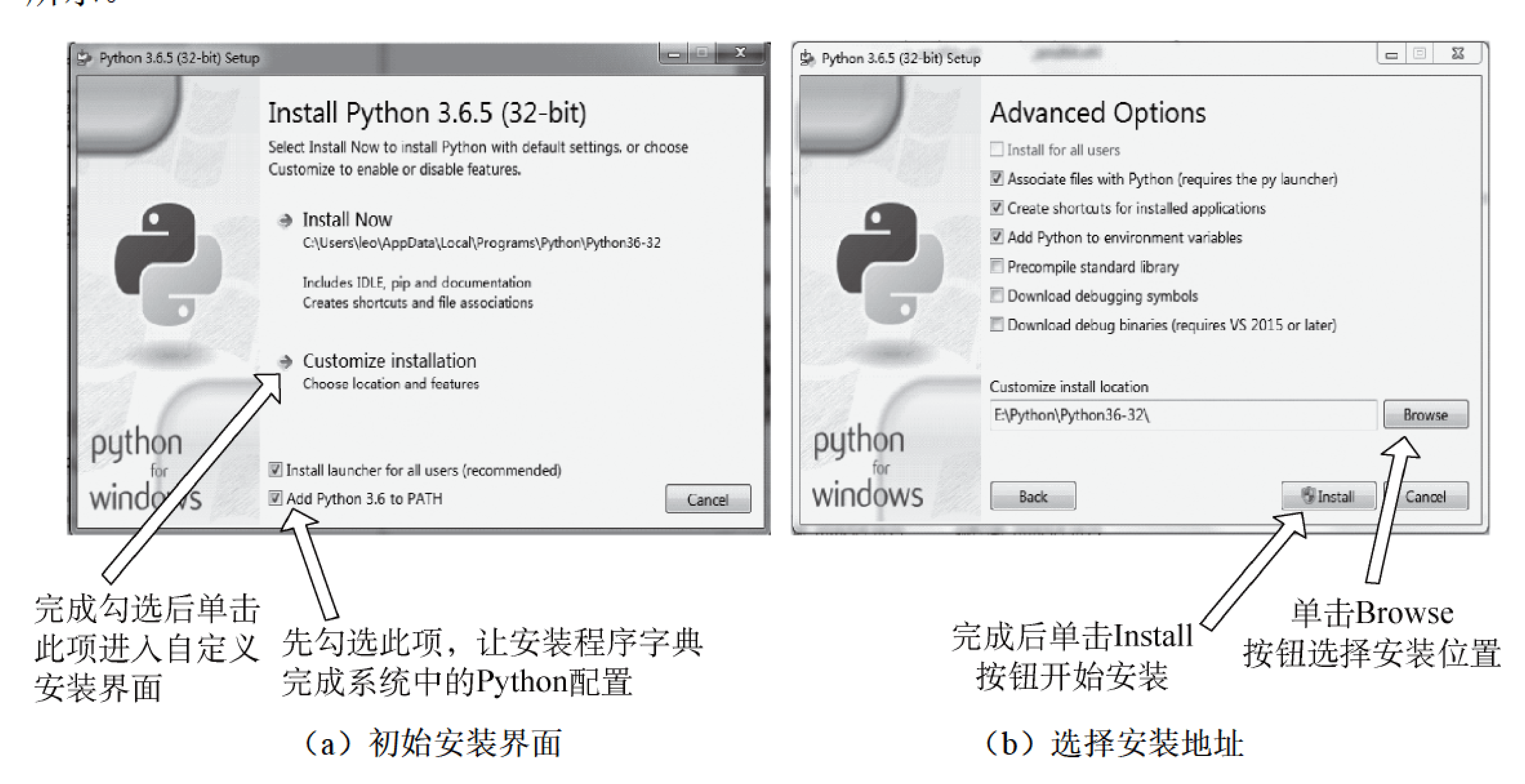


图 1.6　下载 Python 安装程序

完成下载后，找到下载的文件 python-3.6.5.exe 并双击打开。安装程序界面如图 1.7 所示。



（a）初始安装界面　　　　（b）选择安装地址

图 1.7　安装过程

安装时要注意的事项已经在图 1.7 中展示，其余的安装过程直接单击 Next 按钮即可，直到安装完成。

1.5.2　安装 PyCharm

首先进入 PyCharm 下载页面，网址为 https://www.jetbrains.com/pycharm/，我们选择 PyCharm 的开源版本，这是免费的，具体下载过程如图 1.8 所示。

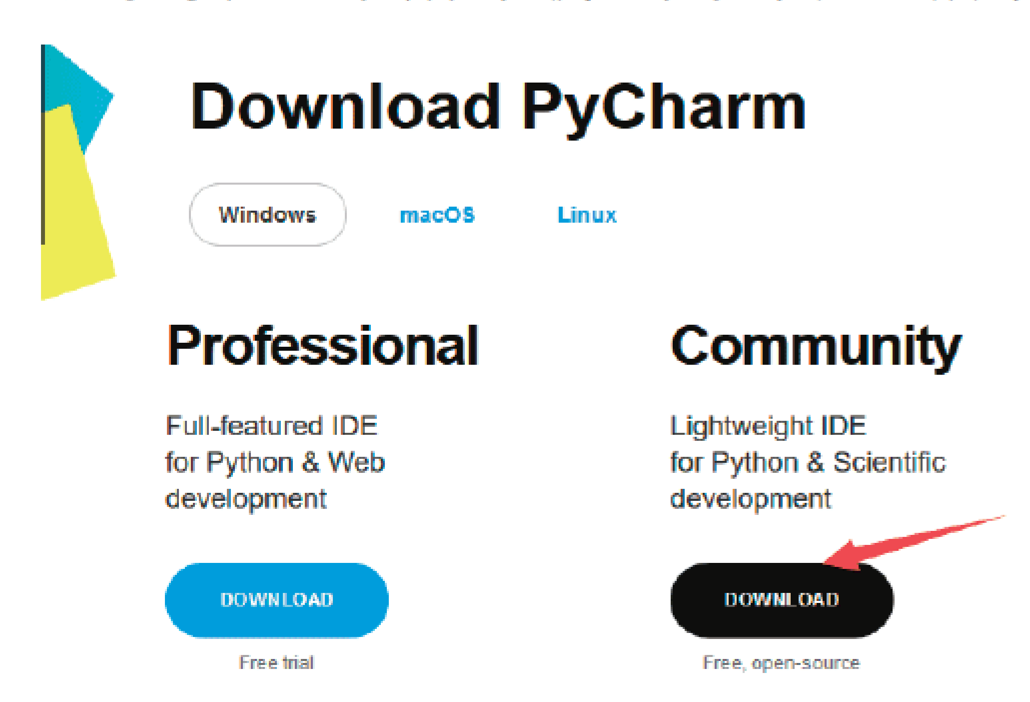


图 1.8　下载 PyCharm 免费版

下载完成后，双击下载的文件 pycharm-community-2018.1.2.exe 开始安装，单击后会出现如图 1.9 所示的安装界面，我们可以一直单击 Next 按钮完成安装，也可以自定义安装路径安装。

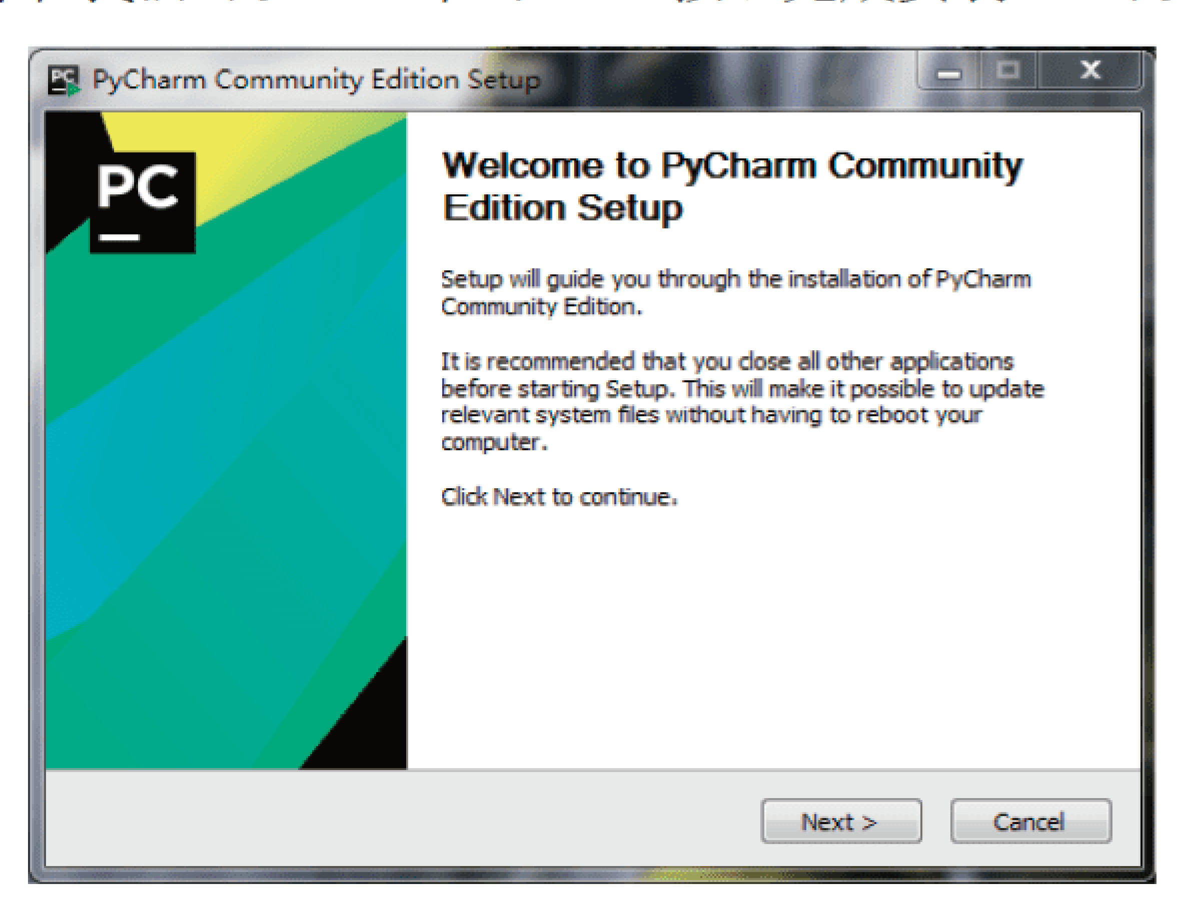


图 1.9　PyCharm 的安装界面

完成 PyCharm 的安装之后，运行 PyCharm 会出现选择界面风格的窗口，完成选择后进入如图 1.10 所示的创建工程界面。

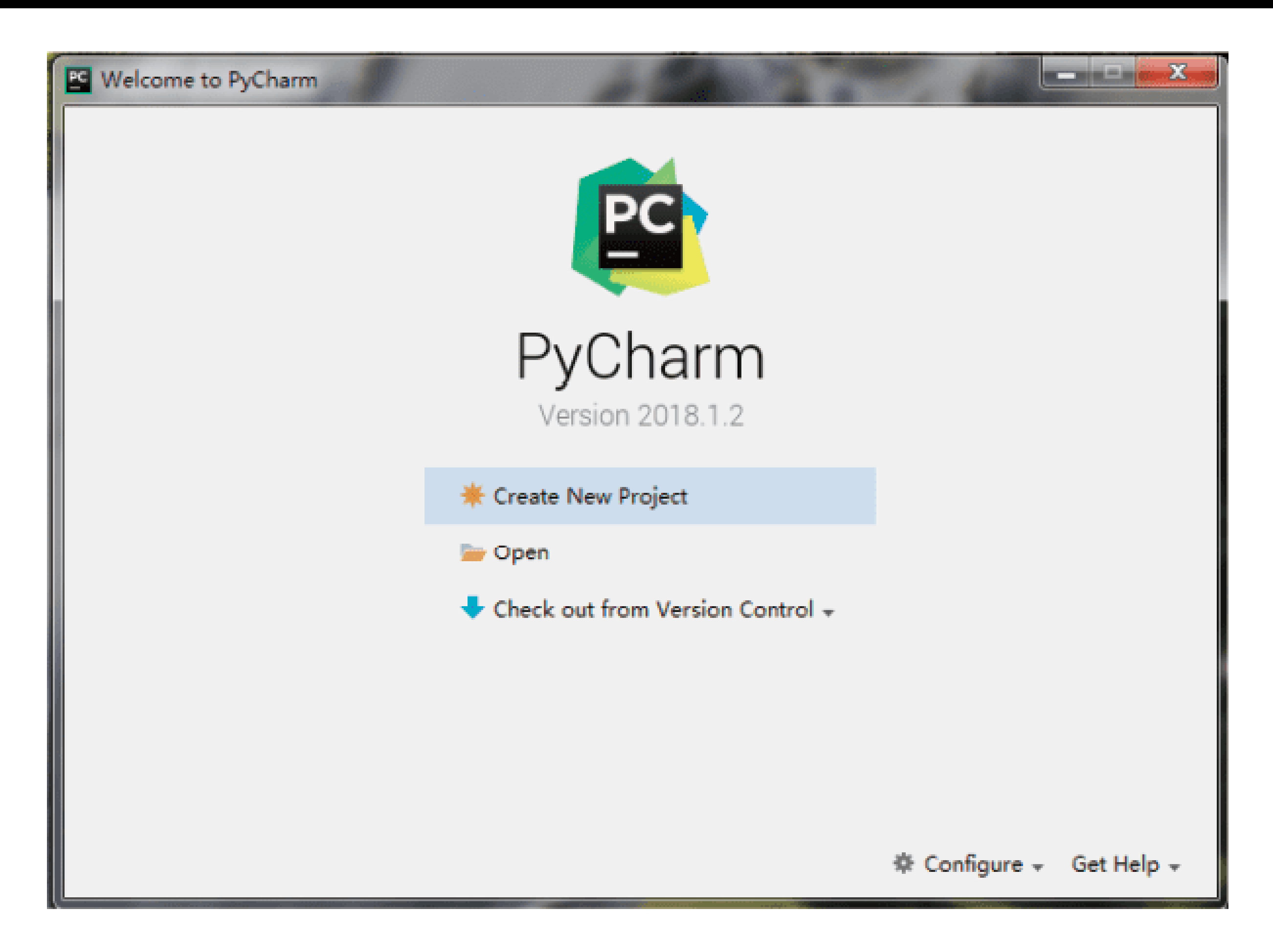


图 1.10　创建工程

在 PyCharm 中的工程表示的是一个项目，它允许我们定义一个或多个 Python 文件。在进入编辑界面之前，先配置工程，具体的操作如图 1.11 所示。

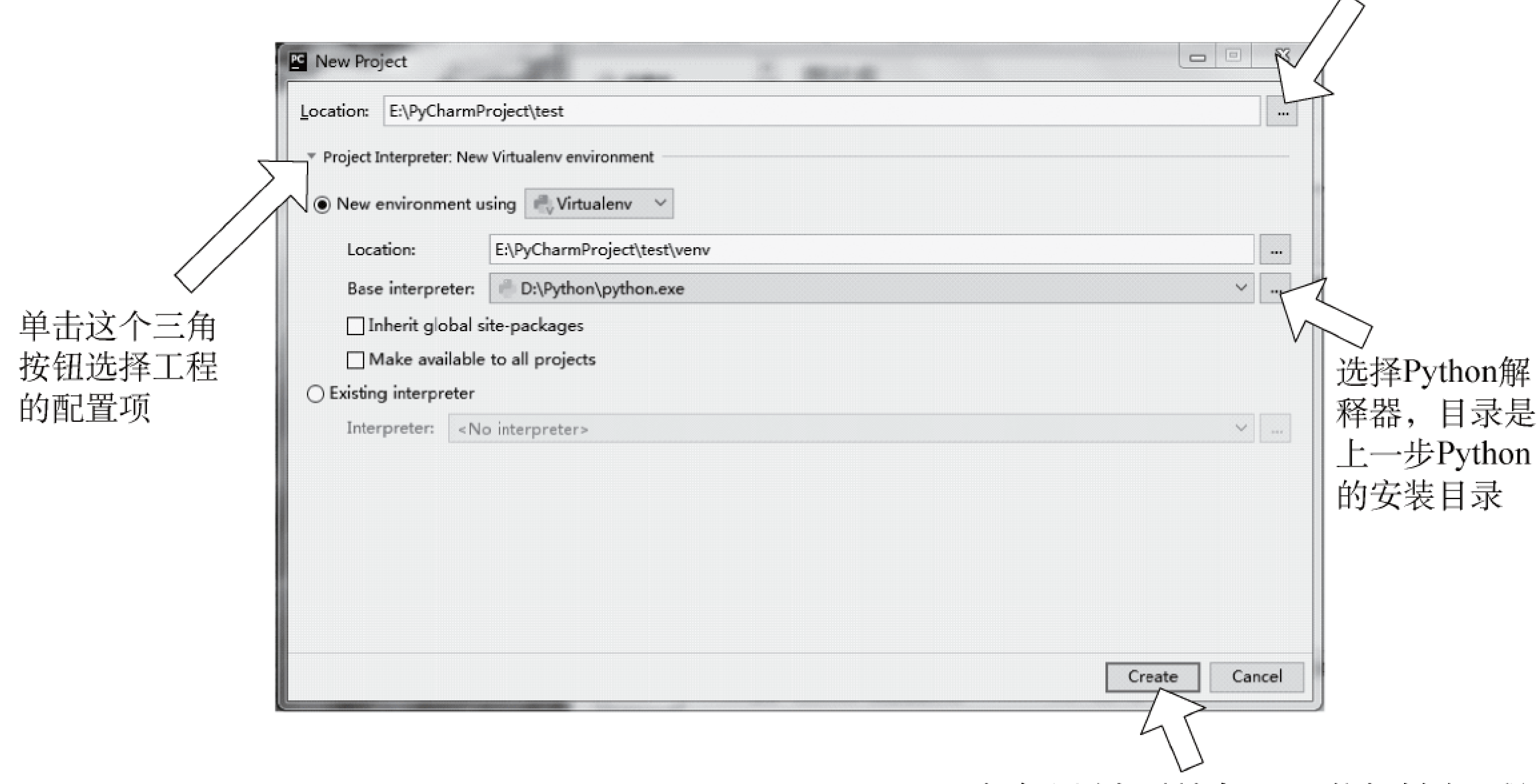


图 1.11　工程配置的具体操作

完成工程创建之后我们再向工程中添加 Python 文件用于程序编写，过程如图 1.12 所示。

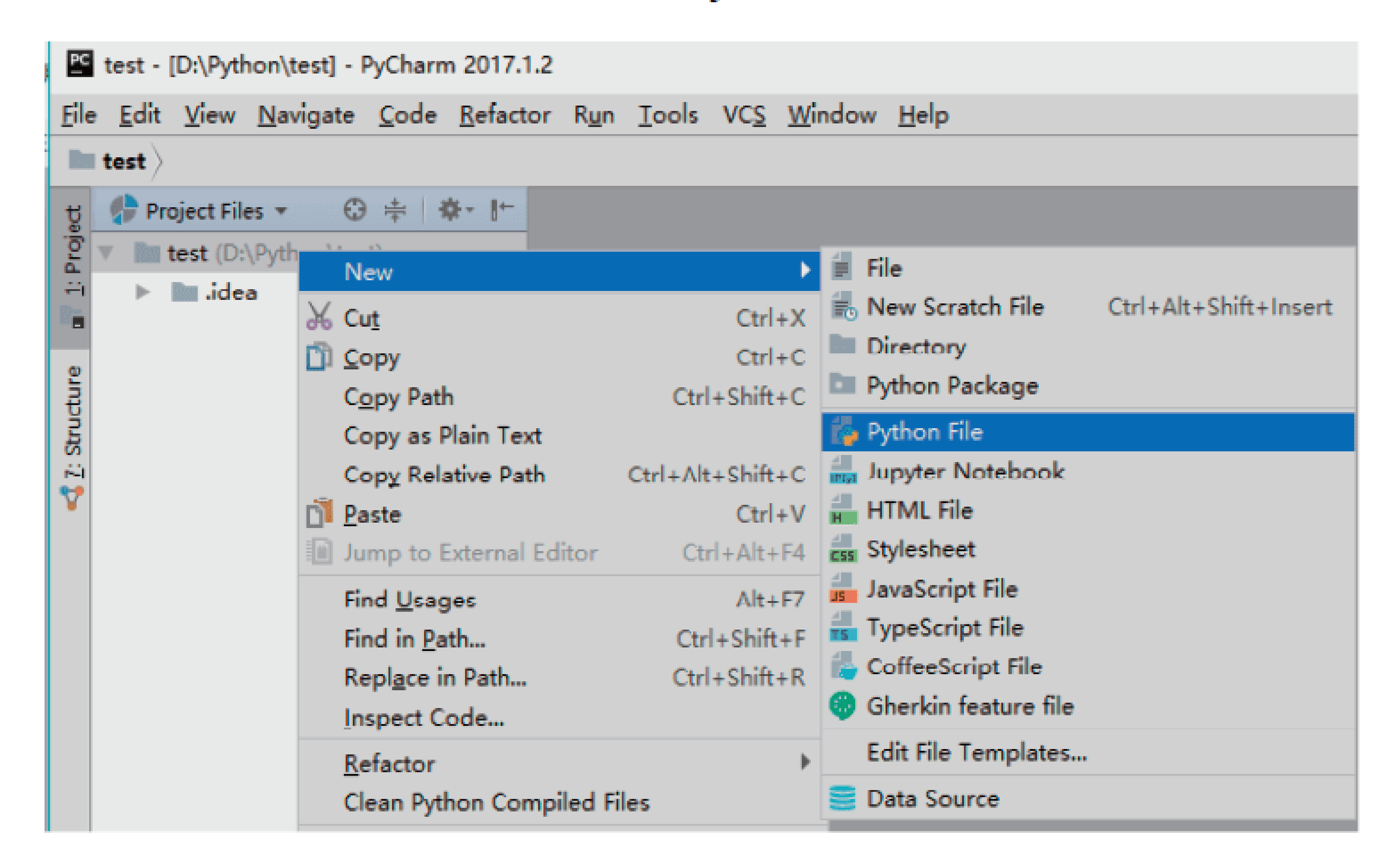


图 1.12　向工程中添加 Python 文件

至此，新的 Python 文件已经创建，下面我们来看看 PyCharm 提供的开发界面，如图 1.13 所示。

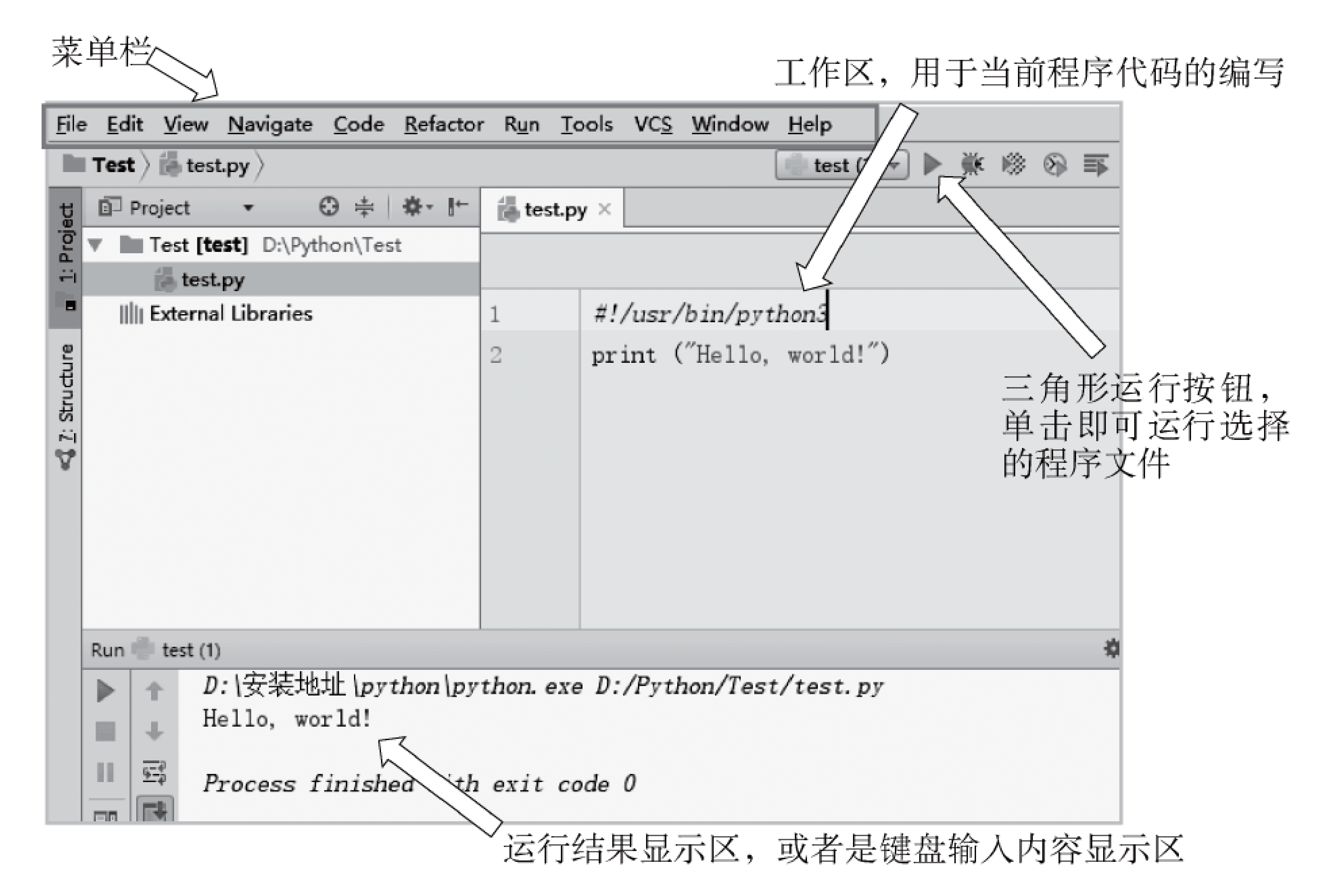


图 1.13　PyCharm 开发界面

图 1.13 中为了演示编码效果，添加了一条 print 语句，它的具体使用方法在后文会讲到。

1.6　Python 编程入门

接下来我们将进入 Python 编程的世界，先来看一些基础知识。

1.6.1　Hello world

Hello world 是很多程序语言运行的第一个结果，被程序员戏称为“经典世界问候语”。

在刚安装好的 PyCharm 中的工作区输入程序 1.1 所示代码，然后单击右上角的运行按钮▶。执行该代码后，会在运行结果显示区显示执行结果 Hello world。

程序 1.1　输出 Hello world：

```
1:    print("Hello world")
```

输出：

```
Hello world
```

分析：

通过以上程序会发现，Python 语言的编写是如此简单，print 函数就是输出函数，我们只需要将要输出的变量或内容放入函数的括号中即可，这里我们将 Hello world 放在引号中作为一个字符串类型，最后输出。

经过上述解释你可能还有些疑惑，不过没有关系，有关输入/输出、函数、字符串等知识在后文都将会提到。此时，PyCharm 的界面如图 1.14 所示。

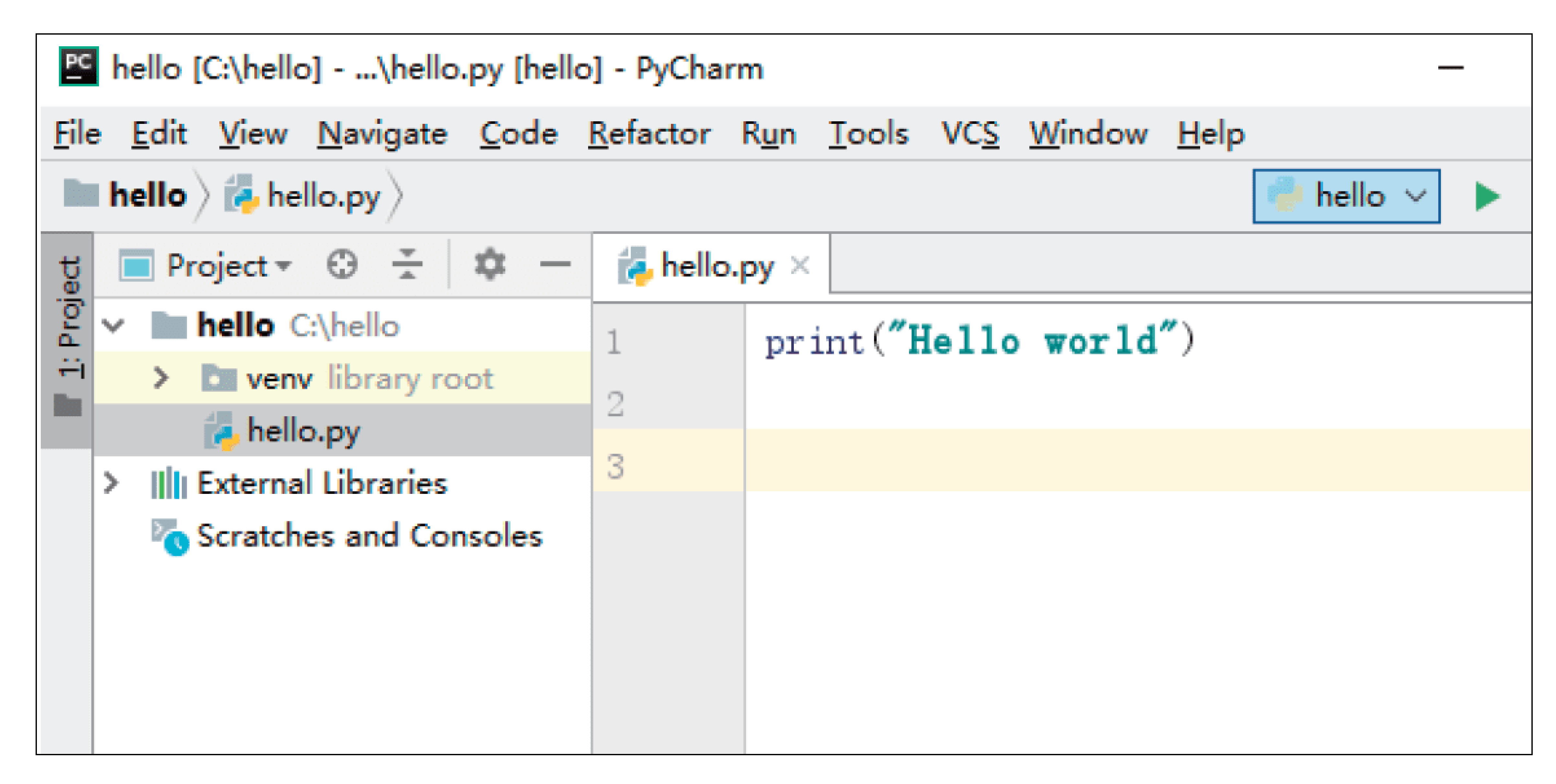


图 1.14　Windows 下的 PyCharm 界面

学习了 Hello world 之后我们再来看一个稍微复杂一点的程序。

程序 1.2：

```
1:    #get your name
2:    name = input("What is your name?\n")
3:    #print Hi + your name
4:    print("Hi,", name)
```

输出：

```
What is your name?
Leo
Hi, Leo
```

分析：

在这里我们将整个程序看成一个工作区，每行语句都有它的功能，程序第 2 行的 name 是该工作区的一部分，它存储了一些信息并且是可以被更改的，我们称它为变量，在对变量命名时要尽量贴合变量的具体使用含义。我们只能在当前工作区使用该变量，因为当前的工作区是它的命名空间，这些在本书后文中都会有介绍。

通过程序输出我们可以看出，程序第 4 行的输出语句将 Hi 和输入内容输出，这是因为第 2 行中使用 input 接收了输入的名字并存在 name 中，第 4 行直接使用变量名就可以调用变量 name 中的内容。在输入时，按 Enter 键表示输入完成。

程序 1.2 看似有 4 行，但是实际上工作的代码只有第 2 行和第 4 行，程序中第 1 行和第 3 行中的内容是以“#”号开头而不起任何作用的注释，它们用于让我们的程序更易懂。

如果上述内容让你有点困惑，这并不是什么问题，因为才刚开始起步，后面将会详细介绍有关内容。

1.6.2　Python 解释器

接下来将会把 Python 的解释器介绍给大家，这样在总体上将对 Python 有一定的认识。

我们现在对于 Python 的认识还只是作为一门编程语言，从程序的实现来看，Python 也是一个解释器类型的软件包。那么，什么是解释器？在这里，解释器是一种让其他程序运行起来的程序。以程序 1.1 为例，我们写下 print("Hello world")之后，Python 解释器读取该语句，按照其中的命令执行，得出结果。也就是说，解释器是夹在代码和计算机硬件之间的软件逻辑层。根据选用的 Python 版本，解释器可以是 C 程序实现，也可以是 Java 程序实现。在这里必须明白的是我们所编写的 Python 代码必须在解释器中运行。有关解释型语言和编译型语言的区别，如表 1.1 所示。

表 1.1　解释型语言和编译型语言的区别

解释型	编译型
执行方式类似于我们日常生活中的“同声翻译”，应用程序源代码一边由相应语言的解释器“翻译”成目标代码（机器语言），一边执行，因此效率比较低，而且不能生成可独立执行的可执行文件，应用程序不能脱离其解释器，但这种方式比较灵活，可以动态地调整、修改应用程序。我们在本书中学习的 Python 属于解释型语言	编译是指在应用源程序执行之前，就将程序源代码“翻译”成目标代码（机器语言），因此其目标程序可以脱离其语言环境独立执行，使用比较方便、效率较高。但应用程序一旦需要修改，必须先修改源代码，再重新编译生成新的目标文件（*.OBJ）才能执行，只有目标文件而没有源代码，修改很不方便。如 C/C++、Java 等都是编译型语言

1.6.3　执行 Python 程序

这个问题作为 Python 解释器的延伸，请大家认真阅读并仔细理解下面的内容，这会建

立起你对 Python 的整体认识。我们从程序编写人员和 Python 解释器两个角度来分别介绍。

还是以程序 1.1 这个简单的程序为例，作为一个程序开发者，一个 Python 程序仅是一个包含 Python 语句的文本文件，程序 1.1 对应的正是 hello.py 文件。对于这个文件，我们只需要告诉 Python 从头到尾执行文件中的语句，也就是说，把我们的逻辑使用 Python 语句表达出来即可，剩下的只需要将其交给 Python，也就是之前说的在 Python 解释器中运行代码。

那么作为 Python，它是如何看待这个问题的呢？

Python 接收到我们的程序之后，将其编译成字节码，之后再将其转发到“虚拟机”中。下面，我们再详细分析一下上述步骤。编译其实是一个简单的翻译步骤，它把每一条语句（就像程序 1.1 中的 print("Hello world")）翻译成一组字节码指令。完成后，字节码发送到 Python 虚拟机（Python Virtual Machine，PVM）上执行。而对于 PVM，我们可以理解为一个迭代运行字节码指令的大循环，一个接一个操作，它是实际运行脚本的组件。执行的图解如图 1.15 所示。

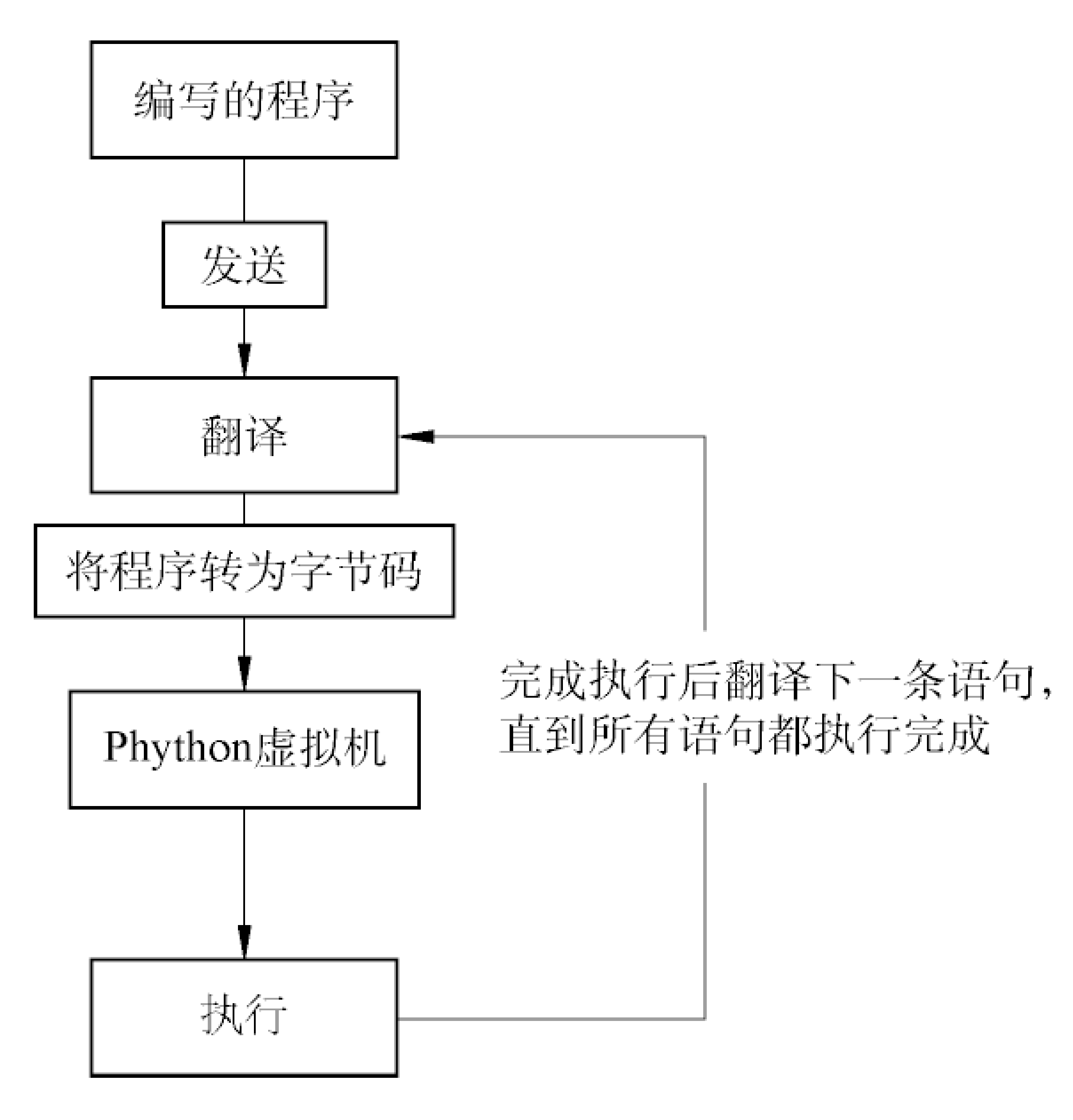


图 1.15　Python 语句执行过程

1.6.4　Python 基础

我们已经通过执行程序 1.1，输出了 Hello world。但这仅仅是 Python 编程的一个开始，

下面介绍 Python 编程的一些基础知识，为以后的编程打下基础。

1. 注释

在“#”号右边的文字并不影响程序的执行结果，它主要是为了解释语句的功能，是写给程序的读者看的笔记。不要小看这个功能，它会让程序的读者更好地理解代码语句。

例如：

```
print('Hello world') #注意到 print 是一个输出函数
```

2. 字面常量

当使用字面常量时，我们用的就是它字面意义上的值或是内容。例如，97、5.29 这样的数字或是 Hello world 这样的文本。它们的值都是不能被改变的，因此它们被称为字面常量。

3. 数字

在 Python 中，数字主要分为两种：整数和浮点数。

整数大家都知道，如 78。而对于浮点数（Float Point Numbers，简写 floats），则有 5.29 或是 78.2E-4（E 表示 10 的幂，这种表示对应 78.2*10^-4）这两种表示方法。

4. 字符串

字符串是字符的序列，字符串也称为 String。在这里多提一句，几乎所有我们编写的 Python 程序都会用到字符串。

（1）声明字符串

单引号。我们用单引号来指定字符串，如'I am a string'或者'this is a string'这种。引号中的内容会按原样保留。

双引号。也许你足够细心已经发现上面'I am a string'中没用 I’m 的缩写，为什么？没错，因为外面是单引号，如果再用缩写的话，解释器会产生符号识别错误。我们使用双引号来避免这种情况，如"I’m a string"这样做就完全可以。它的工作机制同单引号一样。

三引号。三引号是使用"""或'''来指定多行字符串，我们可以在三引号之间自由地使用单引号和双引号。三引号也可以用来做多行注释，需要注意的是，必须成对使用。

例如：

```
'''  This's the first line.
          Second.
          Third "end".
'''
```

注意：

一旦指定了字符串，我们就不能再改变它。

（2）字符串的使用

构建字符串。有时我们要从其他信息中构建字符串。推荐使用 format 方法来构建，具体做法会在后文中讲解。

转义序列。前面在介绍双引号时提到缩写的情况若还要使用单引号会报错，这里我们提出问题：为什么报错？这是因为使用单引号在写 I'm 时，重复的单引号会让 Python 对于何处是字符串的开始、何处是结束感到困惑。所以，我们必须指定这个单引号不代表字符串的结尾。这就是转义序列的作用。还是缩写那个例子，我们可以通过“\”来定义单引号，对应字符串为'I\'m a string'。

如果想要指定一串新行字符串该如何操作？当然了，我们可以用之前提到的三引号。或者，使用表示新的一行的转义序列“\n”来表示新的一行。例如，下面这行字符串：

```
'This is the first line\n Second line'
```

还有许多转义序列，在今后的程序中遇到时再详细介绍。

5. 标准数据类型

变量可以将各种形式的值保存为不同的数据类型。接下来先来介绍 Python 中的标准数据类型。Python 中支持 5 种标准的数据类型：数字、字符串、列表、元组和字典。前面讲了数字和字符串，剩下的 3 种我们以后再细谈。

6. 运算符

下面来详细谈谈运算符，它在实际编程中是很常用的。先举个简单的例子 5+9=14，这个式子对我们早已不是什么难题，但是通过它来引出：式子中的“+”称为运算符，4 和 5 称为操作数，是不是和笔算时很相似？

Python 中有很多运算符，这里先介绍算术、比较、逻辑以及三元运算符。

（1）算术运算符。算术运算符会根据运算符做出相应操作并返回结果，具体如表 1.2 所示。

表 1.2　算术运算符

运算符	描述	例子
+	两个操作数相加	40 + 21 = 61
–	两个操作数相减	85 – 12 = 73
*	连个操作数相乘	9 * 3 = 27
/	两个操作数相除	9 / 3 = 3
%	返回除法的余数	10 % 3 = 1
**	幂	10**3 = 1000
//	返回商的整数部分	9 // 4 = 2

（2）比较运算符。在详细列出比较运算符之前，先提示一下，就像算数运算符会返回结果一样，比较运算符会返回 True/False（或是返回 1/0，1 表示真，0 表示假。这分别与 True/False 等价）作为比较结果。比较运算符具体如表 1.3 所示。

表 1.3　比较运算符

运算符	描述	例子（a = 30, b = 15）
==	比较两个对象是否相等	a == b（返回 False）
!=	比较两个对象是否不等	a != b（返回 True）
<>	比较两个对象是否不等	a <> b（返回 True）
>	大于，x>y 返回 x 是否大于 y	a > b（返回 True）
<	小于，x<y 返回 x 是否小于 y	a < b（返回 False）
>=	大于等于	a >= b（返回 True）
<=	小于等于	a <= b（返回 False）

（3）逻辑运算符。Python 的逻辑运算符语法比较简单明了，和高中的数学教材中逻辑部分的内容是相同的，具体如表 1.4 所示。

表 1.4　逻辑运算符

运算符	表达式	描述	例子（a = 30, b = 15）
and	x and y	布尔“与”，如果 x 为 False，x and y 返回 False，否则返回 y 的计算值	a and b（返回 15）
or	x or y	布尔“或”，如果 x 是非 0，返回 x 的值，否则返回 y 的计算值	a or b（返回 30）
not	not x	布尔“非”，如果 x 为 True，返回 False。如果 x 为 False，返回 True	not(a and b)（返回 False）

（4）三元运算符。Python 中唯一的一个三元运算符如表 1.5 所示。

表 1.5　三元运算符

表达式	描述	例子
on_True if expression else on_false	为 True 时的结果 if 判定条件 else 为 False 时的结果	1 if 5>3 else 0（输出 1）

思考：

print(5>=3 if 3 and not 7 else 1)这条语句会输出什么？

1.7　变量及其赋值

上文中提到，数字、字符串都是用它们字面意义上的值或是内容，这叫作字面常量。但是如果一直使用字面常量会让人感到过于死板，更不足以应对复杂的情况。我们要做的是操作所存储的信息，这时便需要引入变量。就像这个名字一样，变量的值是可以变化的，Python 允许我们用变量来存储任何东西。因为要操控它，所以我们要为它们命名并通过这个名字访问变量。

在 Python 中变量可以存储各种形式的值并保存为不同的数据类型（若你之前接触过 C/C++等其他程序设计语言，在这里注意了，Python 中不需要变量声明）。

在 Python 中，每个变量在使用前都必须赋值。变量赋值之后会被自动创建。在程序中，我们使用等号来给变量赋值。等号左边是变量名，右边是存储在变量中的值。例如：

```
name = "Leo"
```

这便是将字符串"Leo"赋值给 name 变量。

1.8　输入与输出

本节将介绍一下输入与输出，这里只是介绍在 PyCharm 的运行结果界面中的输入与输出，有关于涉及文件部分的输入与输出会在后文中介绍。下面先看看输入与输出的流程。带有输入与输出的程序执行过程如图 1.16 所示。

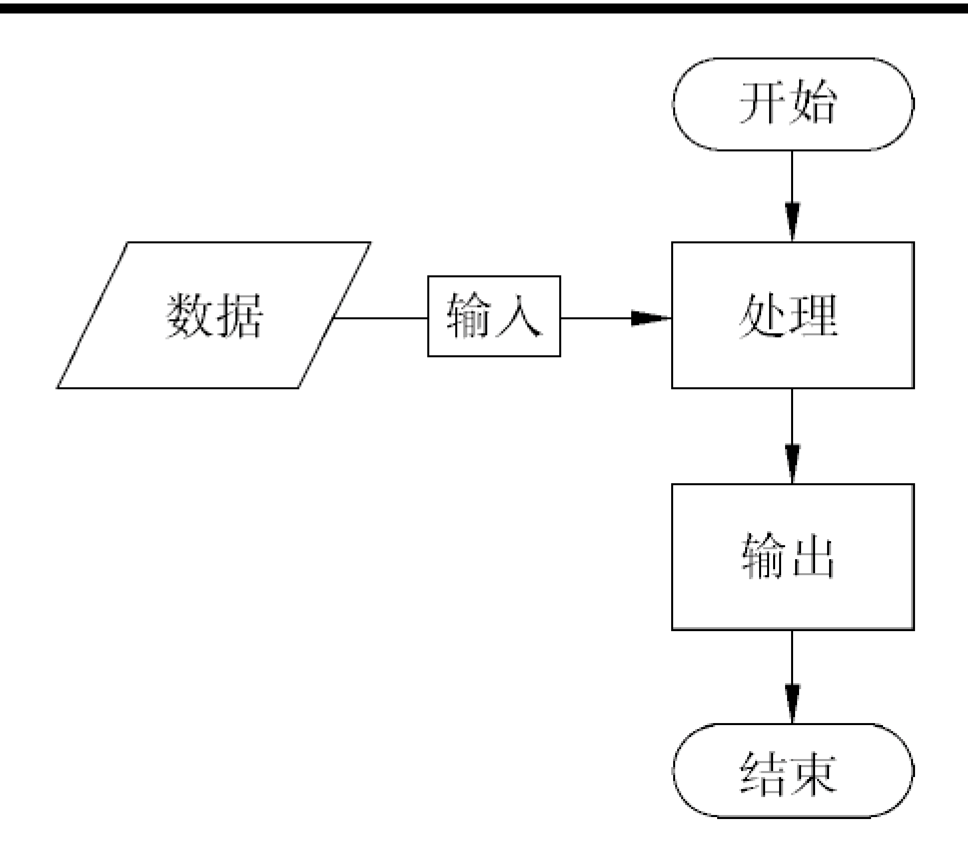


图 1.16　带有输入与输出的程序执行过程

1.8.1　输出

本章程序 1.1 用一个 print 函数输出 Hello world，在这里我们详细谈谈输出语句。不知道你有没有发现，print 函数总是会以一个不可见的“新一行”字符（\n）结尾，因此重复调用 print 将会在相互独立的一行中分别打印。为了防止这一现象，Python 中的 print 函数可以通过 end 指定除换行以外的其他结尾。例如，可以通过 end 指定以空白结尾：

```
print('a', end='')
print('b', end='')
```

输出：

```
ab
```

1.8.2　输入

对应输出，Python 也提供了一种输入机制，也就是 input 函数，它可以接收我们从键盘输入的内容，把值存入变量中。另外，input 函数的参数可以设置为当用户进行输入之前要显示的信息，如 input("please input some num.")这种形式。

一旦用户按下 Enter 键，表示输入完成并退出函数。

注意：

程序接收到输入的内容是字符型的。

下面使用一个程序来熟悉一下我们学过的一些语法。这是一个编程实现简单计算器的

程序，程序接收两个操作数和一个操作符并将计算结果输出。

程序 1.3　简单的计算器：

```
1:   x = int(input("please input the first operand: "))
2:   y = int(input("please input the second operand: "))
3:   character = input("please input the instraction character: ")
4:
5:   result = x+y if character == "+" else None
6:   result = x-y if character == "-" else result
7:   result = x*y if character == "*" else result
8:   result = x/y if character == "/" else result
9:
10:  print("the result is : {0}".format(result))
```

输出：

```
please input the first operand: 9
please input the second operand: 5
please input the instraction character: /
the result is : 1.8
```

分析：

首先，程序 1.3 仅仅用于演示，它是一个很不完善的程序，如果你的操作数输入不是数字的话程序将会崩溃。

程序前 3 行用于接收用户的输入，每次输入完成按 Enter 键结束输入，输出前 3 行为用户从键盘输入要运算的数值，输入的数值字体为斜体，如程序 1.3 中的输出结果所示。由于操作数是整型，而 input 函数对接收输入内容返回的是字符串类型，因此我们用一个 int 函数将其类型转换成整型。程序的 5～8 行使用了三元运算符来判断操作符并选出对应操作符的具体操作。

最后，注意第 10 行的输出语句，这就是我们上文中提到的使用 format 方法来从其他信息中构建字符串。在这个字符串中的{0}将会被 format 的第一个参数代替。若是有两个参数如何做呢？例如，在程序输出结果中我们要输出操作数 x，y：

```
print("the operand is {0} and {1}".format(x, y))
```

注意：

使用 format 方法明了而且不容易出错，但是一定记住 Python 是从 0 开始计数，即索引中的第一位是 0，第二位是 1，以此类推。

警告：

缩进（各行开头的空白区，用 4 个空格表示）在 Python 中非常重要，它用于确定语句的分组。放置在一起的语句必须拥有相同的缩进。每一组这样的语句被称为“块”。在本书的后文中会了解块这一概念的重要性。如程序 1.3 的前 3 行如果写成：

```
1:  x = int(input("please input the first operand: "))
2:      y = int(input("please input the second operand: "))
3:  character = input("please input the instraction character: ")
```

这时程序会报错：

```
y = int(input("please input the second operand: "))
    ^
IndentationError: unexpected indent
```

Python 指出的错误信息告诉我们这个程序的语法是无效的。有关使用新块的情况后文会提到。如果我们使用 PyCharm 编写程序的话，它会自动处理缩进。

1.9 趣味练习

之前的学习都是先介绍知识点，再使用例题强化对知识点的认知，总体风格偏向于严肃的课堂风，但是在本节以及后续章中的“趣味练习”会让我们放松，慢慢体会并尽可能地享受编写程序过程中的乐趣。

接下来使用 Python 来模拟一个我们与机器的交互，先来看程序 1.4。

程序 1.4：

```
name = input("What is your name?")
age = input("How old are you?")
sex = input("What is your sex?")
print("Now I know you,{0},{1} years old {2}. \n".format(name, age, sex))
print("Thank you for reading this book.")
```

输出：

```
What is your name?Leo
How old are you?18
What is your sex?boy
```

```
Now I know you,Leo,18 years old boy.

Thank you for reading this book.
```

点睛：

学习完本章前面的内容再来看程序 1.4 是不是有一种水到渠成的感觉，在这里只强调一个地方，在使用 format 构造字符串时如果只有一个值，在引号中可以直接使用{}而不必使用{0}这种形式。

1.10　总　　结

在本章中，我们学习了如何在 Ubuntu 和 Windows 下搭建 Python 的开发环境，Python 入门以及如何编写和执行第一个 Python 程序。本章还介绍了 Python 的发展历程，Python 语言的优缺点，以及 Python 内部的解释机制。在接下来的章中我们会深入学习 Python 的具体语法。

1.11　练　　习

（1）列举使用 Python 的几个理由。

（2）分阶段详细写出程序运行时 Python 内部的执行过程。

（3）下面的程序存在着什么样的错误？

```
print('''
this is the first line
this is the second line
''')
print("hello Python")
```

（4）使用算数运算符完善程序 1.3 中的计算器，让它支持更多的计算方式。

第 2 章　基本数据类型

本章将会涉及一些常用的 Python 数据类型以及在中学数学课上学习过的数据类型的表达方法。完成本章的学习之后，需要掌握：

- 分数的定义方法以及处理方式。
- 关于复数的具体操作。
- 灵活使用字符串。
- 布尔型数据和逻辑运算符的使用。

2.1　分数和复数的表示

先来谈谈分数和复数，这都是我们在数学上学习或是将要学习的知识。通过本章的学习后，我们可以利用 Python 处理更多的操作。

2.1.1　分数

Python 可以处理分数，但是要使用一个分数模块。我们对模块的概念可能还很模糊，但在后面会正式认识它。现在，我们只需把它看作一个黑盒，知道它是可以处理分数的工具即可。

接下来，程序 2.1 将分数的定义以及各种使用方法列出，我们对照程序分析学习具体的方法。

程序 2.1：

```
from fractions import Fraction

f = Fraction(1, 2)
print("print Fraction(1, 2) : {0}".format(f))

f = Fraction(2.5)
```

```
7:    print("print Fraction(2.5) : {0}".format(f))
8:
9:    f = Fraction("2.5")
10:   print("print Fraction('2.5') : {0}".format(f))
11:
12:   f = Fraction(122, 368)
13:   print("print Fraction(122, 368) : {0}".format(f))
14:
15:   f = Fraction(1, 2) + Fraction(3, 8)
16:   print("print Fraction(1, 2) + Fraction(3, 8) : {0}".format(f))
17:
18:   f = Fraction(7, 8) + 3
19:   print("print Fraction(7, 8) + 3 : {0}".format(f))
20:   f = Fraction(7, 8) + 3.0
21:   print("print Fraction(7, 8) + 3.0 : {0}".format(f))
22:
23:   f = Fraction(1, 2) * Fraction(2, 8)
24:   print("print Fraction(1, 2) * Fraction(2, 8) : {0}".format(f))
25:
26:   f = Fraction(1, 2)
27:   print("the numerator of f is : {0} and denominator is : {1}".format(f.numerator, f.denominator))
```

输出：

```
print Fraction(1, 2) : 1/2
print Fraction(2.5) : 5/2
print Fraction('2.5') : 5/2
print Fraction(122, 368) : 61/184
print Fraction(1, 2) + Fraction(3, 8) : 7/8
print Fraction(7, 8) + 3 : 31/8
print Fraction(7, 8) + 3.0 : 3.875
print Fraction(1, 2) * Fraction(2, 8) : 1/8
the numerator of f is : 1 and denominator is : 2
```

分析：

程序的第 1 行将 Fraction 引入。程序的 3、6、9 行对应着不同的分数指定方法，第 3 行中我们使用 Fraction(1, 2)的方式，第 6 行将浮点数 2.5 作为参数，第 9 行甚至将一个字符串作为参数传入。但是，看看前 3 行输出结果：这 3 种分数声明方式都是正确的，都会返回一个分数，即我们可以根据实际的需求选用任意一种分数声明方式。

程序的第 12 行我们指定了分数 122 / 368，这是一个可约分的分数，通过第 4 行的输出可以看到约分已经完成。

程序的第 15、18、20 行都是简单的加法运算，但输出的结果却不同。第 15 行，两个 Fraction 对象相加时，输出同之前的一样。第 18 行，Fraction 对象和一个整型数相加，输出也同之前一样。但是，我们来看看第 20 行，当 Fraction 对象和一个浮点数相加时，输出的结果却变成了浮点数。如果程序对输出要求很严格的话，应该想好需要使用什么类型。

最后，来看看程序的第 23 行和第 27 行。第 23 行是两个 Fraction 对象相乘返回一个分数，它们不是简单地互乘，相乘的结果会自动约分之后再输出。第 27 行使用了 Fraction 中的 numerator 和 denominator 两个属性，它们分别是分数的分子和分母。

关于 Python 中分数的操作先介绍到这里。后续若有其他关于分数的操作我们再根据具体的程序进行分析。

2.1.2　复数

到目前为止，我们接触到的都是实数，但是不管是在学校的课程（人教版课本是高中数学选修 2—2。若你还没有学到复数这一概念，可以先跳过这节，或是查阅数学教材再来阅读。我们不在这里细谈数学概念）中还是在实际的工程操作中，这都是远远不够的。

Python 为我们提供了复数以及关于复数的操作。注意，在数学课上通常用 i 来定义复数的虚部，但是在 Python 中使用的是 j。例如，在笔算时通常会写 2+3i 这种形式，但是，在 Python 中我们要输入 2+3j。接下来用程序 2.2 来看看复数的具体操作。

程序 2.2：

```
a = 2 + 3j
b = complex(4, 9)
print(type(a))
print(type(b))

print(a + b)
print(a - b)
print(a * b)
print(a / b)

print("the real of a is : {0}, imag is {1}".format(a.real, a.imag))
```

```
12:   print("the conjugate of a is : {0}".format(a.conjugate()))
13:   print("the magnitude of a is : {0}".format(abs(a)))
```

输出：

```
<class 'complex'>
<class 'complex'>
 (6+12j)
 (-2-6j)
 (-19+30j)
 (0.36082474226804123-0.061855670103092786j)
the real of a is : 2.0, imag is 3.0
the conjugate of a is : (2-3j)
the magnitude of a is : 3.6055512754639896
```

分析：

程序的第 1 行和第 2 行使用了两种方法指定复数，第一种使用了我们很熟悉的方法，但是一定注意 Python 中虚部不是用 i 表示。第二种方法是使用 complex 函数，它的第一个参数是实部，第二个参数是虚部，第 2 行语句的结果同 b = 4 + 9j 一样。第 3～4 行是将它们的类型输出，可以看到输出结果中第 1 行和第 2 行是 a 和 b 的类型。关于输出中的 class 关键字在这里先不细谈。

程序的第 6～9 行对应了复数的加减乘除运算，这个相比较笔算而言就很有优势了。我们可以将较为复杂的步骤交给 Python 去做。输出结果中的第 3～6 行是运算结果。

程序的第 11 行分别输出了复数的实部与虚部，这里我们要注意的是输出结果中的第 7 行，使用 real 和 imag 属性输出的是浮点型的数。

程序的第 12 行，我们直接使用 conjugate 方法来输出 a 的共轭复数，结果在输出的第 8 行。

最后，程序的第 13 行，我们使用 abs 方法来计算复数的大小，这和语句“(a.real ** 2 + a.imag **2) ** 0.5”的运算结果是一样的。

2.2　字　符　串

在第 1 章的 Python 基础中已经简单地介绍过字符串的定义方式以及一些注意事项，接下来我们用程序 2.3 来介绍关于字符串的具体操作。

程序 2.3：

```
1:    str = "hello world"
2:
3:    print(str)
4:    print(str[0])
5:    print(str[3:7])
6:    print(str[1:])
7:    print(str * 4)
8:    print(str + " type : String")
```

输出：

```
hello world
h
lo w
ello world
hello worldhello worldhello worldhello world
hello world type : String
```

分析：

在程序的第 1 行中我们看到了很熟悉的字符串的定义，第 3 行是对整个字符串进行打印输出，就像输出结果里的第 1 行显示的那样。下面重点解析第 4～8 行代码。

在 Python 中有两种字符串列表取值顺序：从左到右索引（见表 2.1）和从右到左索引（见表 2.2）。当使用从左到右索引时，默认是从 0 开始，索引值为 0 时表示字符串的第一个字符，以此类推，从左到右依次加 1，最大范围是字符串长度减 1。当使用从右到左索引时，默认是从−1 开始，索引值−1 即表示最后一个字符，从右到左依次减 1，最大范围是字符串的开头。

表 2.1　从左到右索引（第 2 行为索引号）

h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

表 2.2　从右到左索引（第 2 行为索引号）

h	e	l	l	o		w	o	r	l	d
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

理解了取值顺序之后，程序的第 4～8 行也就不难理解了。程序的第 4 行只输出了字符串的第 1 个字符，第 5 行语句的意思是索引值从第 3 个字符开始到第 6 个字符（注意不是到第 7 个字符，str[x, y]表示的是字符从索引值为 x 的字符开始到索引值为 y−1 的字符结束）。程序的第 6 行输出索引值为 1 开始的字符及其往后的整个字符串，即输出结果中第 4 行所示的 ello world。程序的第 7 行表示将 str 的内容输出 4 次。程序的第 8 行使用“+”号将两个字符串拼接成一个连续的字符串。

2.3　布　尔　型

前文已经提到过布尔类型的数据，布尔型只有 True 和 False 两种值。接下来，可以通过程序 2.4 来看看到底什么是 True，什么是 False。

程序 2.4：

```
1:    print(True)
2:    print(False)
3:
4:    print(True and "second")
5:    print(False and "second")
6:
7:    print(True or "second")
```

输出：

```
True
False
second
False
True
```

分析：

程序中的第 1 行和第 2 行将 True 和 False 输出，可以发现它们就是 True 和 False。第 4、5 行用到了第 1 章学到的 and 运算符，从它们的输出结果可以看出 and 运算符的特性。第 4 行的输出结果是一个字符串 second，为什么结果不是布尔值？这是因为 Python 将 0、空字符" "和 None 看成 False，而其他数值和非空字符串都看成 True。

再看看第 5 行的输出结果，它只是输出了 False，这是因为 and 的“短路”特性，and 发现它的第一段是 False 就不会再往后算而是直接返回 False。接下来再看看第 7 行的输出结果，没错，or 也有“短路”特性，or 发现第一段为真之后就不会继续计算而是直接输出 True。

2.4　趣 味 练 习

本章的趣味练习将使用一个模拟书店清单的程序来对买书以及打折的结算过程进行模拟，先来看程序 2.5。

程序 2.5：

```
1:  from fractions import Fraction
2:  number = eval(input("How many books do you want?"))
3:  cost = eval(input("How much does each book cost?"))
4:  total = number * cost
5:  discount = Fraction(3/4)
6:  finally_total = total * discount
7:
8:  print("The cost of these books is {}".format(total))
9:  print("The discount of the activity is {}".format(discount))
10: print("The reality cost of these books is {}".format(finally_total))
11: print("The activity has saved money: {}".format(total - finally_total))
```

输出：

```
How many book do you want?50
How much does each book cost?30
The cost of these books is 1500
The discount of the activity is 3/4
The reality cost of these books is 1125
The activity has saved money: 375
```

点睛：

程序总体上比较清晰，是一个较为简单的单流程程序，在这里要说的是出现在第 2 行和第 3 行的 eval 函数，它的功能是将输入的字符串当成有效的表达式来求值并返回计算结果。

在使用 input 接收输入的时候，返回的输入内容是一个字符串类型，它不能用于后续的计算，在之前的程序中我们使用的是 int 或是 float 对其进行类型转换，但是在这里我们使用的是 eval 函数，因为它十分强大，它的功能也不仅限在本章的趣味练习中使用的那样，关于它在后文还会提到。

2.5　总　结

通过本章的学习会发现，在 Python 中实现一些数学操作也是很方便的，只需要调用某些方法就可以完成。本章讲解了分数和复数、字符型和布尔型的数据类型以及相关的操作，还有一些本章中没有提到的数据类型，后文会具体介绍。

2.6　练　习

（1）写出语句 print(Fraction(9, 5) + 0.2)的运行结果。

（2）编写 Python 程序输出 $\frac{-1-3i}{1-i}$ 的结果。

（3）写出下面程序的运行结果。

```
string = "This is a String"
print(string[1:9])
print(string[-7:-1])
```

（4）分析 True and "second" or "third" 的计算结果是什么，为什么？

第 3 章　Python 的流程控制

本章开始学习 Python 的条件控制语句和循环控制语句，通过本章的学习，需要掌握：

- 熟练使用 Python 中流程控制语句。
- 使用条件语句来处理特殊情形。
- 根据特定的条件改变程序的行为。
- 使用适当的循环语句处理具体情况。

3.1　条件控制语句

Python 中的条件控制语句主要是围绕 if 语句展开的，下面将根据具体内容详细展开。

3.1.1　理解 Python 中的条件控制语句

条件控制语句是通过条件表达式的执行结果来决定后续执行代码的，执行的一般流程如图 3.1 所示。

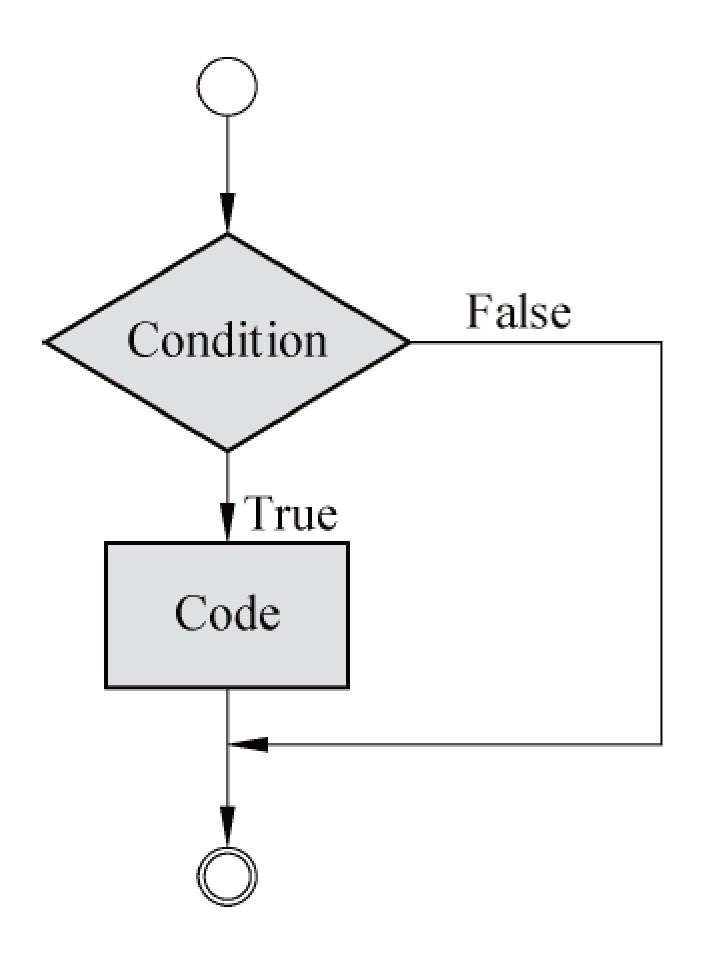


图 3.1　条件控制语句流程图

对于条件表达式，Python 指定非 0 和非空值为 True，0 或 Null 为 False。接下来看看具体的条件控制语句。

3.1.2　if…else 语句

具体语法：

```
if expression:
        expr_true_suite
else:
        expr_false_suite
```

这是一个很好理解的语句，它由 4 部分组成：关键字本身，用于判断结果真假的条件表达式（expression），条件为真（True）时执行的代码块以及条件为假（False）时执行的代码块。

提示：

（1）多重条件表达式的情况：正如字面意思所说，多重条件表达式就是情况复杂以至于一条 if 语句的执行是由很多因素决定的。我们可以使用逻辑运算符 and、or 以及 not 来组合条件表达式。

（2）if…else 结构中的 else 部分是可选的。有时候我们只需要一个判断成功时的操作，而不需要考虑如果判断不成功的情形，即没有 else 及其之后的语句。这在实际的程序编写时也是很常用的。

注意：

尽管 Python 使用的是强制代码正确对齐，这使得在程序中出现不匹配 else 是不可能的，但是你一定要想好 else 是属于哪个 if，因为要改正程序中的这类错误是很费精力的。

我们都知道在同条件的环境下使用催化剂会加快化学反应速度，在这里仅仅抽取化学实验中是否添加催化剂这个条件，来判断化学反应的速度等级。具体程序如程序 3.1 所示。

程序 3.1　判断化学反应速度等级：

```
catalyst = input("Adding catalyst? (y/n)")

if catalyst == 'y':
```

```
4:          print("The speed grade is rapid.")
5:      else:
6:          print("The speed grade is normal.")
```

输出：

```
Adding catalyst? (y/n) : y
The speed grade is rapid.
```

分析：

程序整体上比较简单，但比较明了地展现了 if…else 语句在程序中是如何使用的。程序根据输入判断具体执行哪条输出语句。如果我们的输入值为'y'的话，经过第 3 行的判断语句，程序会转入执行第 4 行。否则的话，程序将直接执行第 6 行。程序流程图如图 3.2 所示。

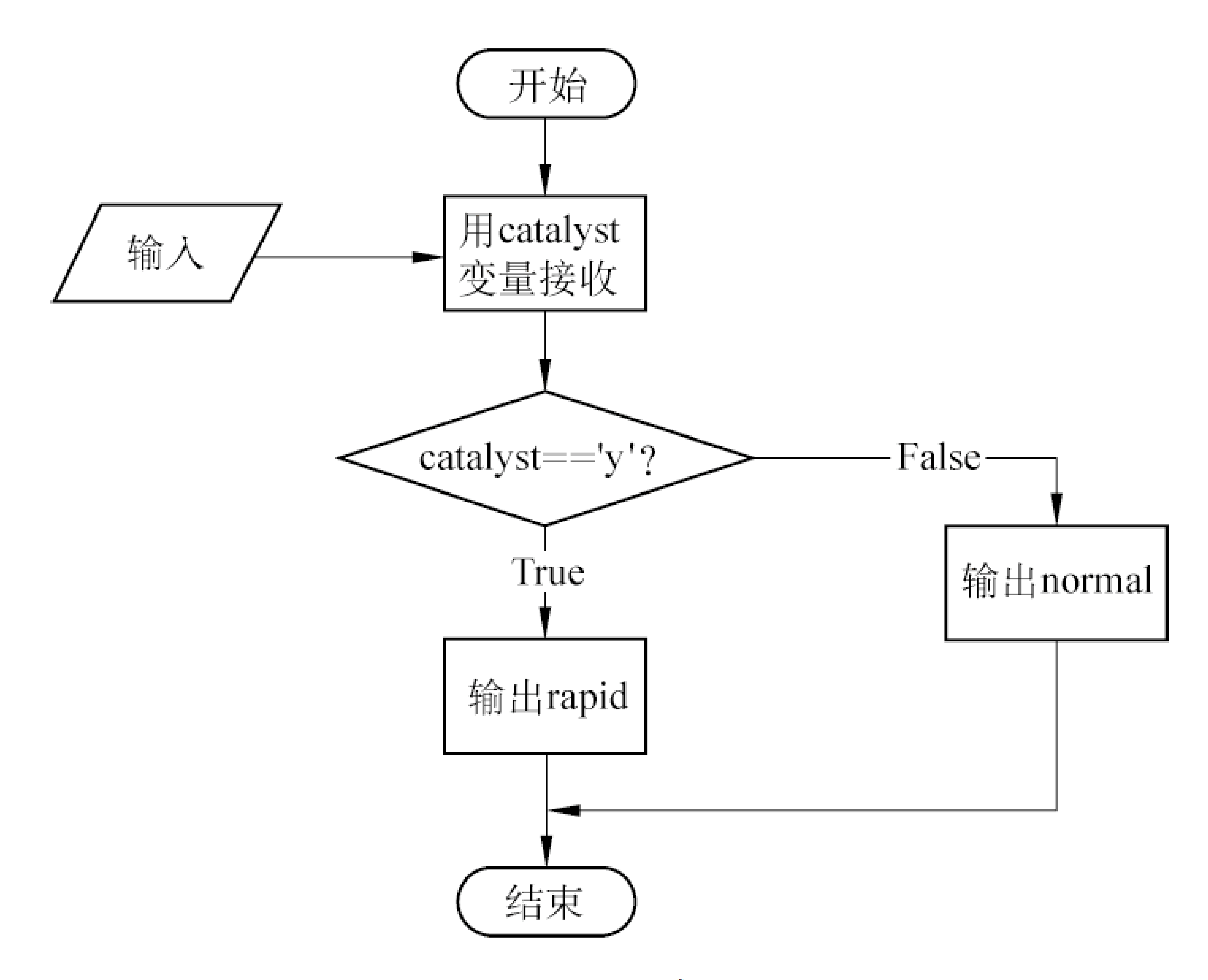


图 3.2　程序流程图

3.1.3　elif 语句

具体语法：

```
if expression1:
    expr1_true_suite
```

```
elif expression2:
    expr2_true_suite
        …
elif expressionN:
    exprN_true_suite
else:
    none_suite
```

elif 即 else if，其中 elif 和 else 一样，声明都是可选。引入这个语句是为了解决有多个条件待选择的情况。假如你要为老师编写一个简单的划分成绩等级的程序，该程序要做到输入分数自动生成对应的等级，分数对应的等级分为不及格、及格、良好以及优秀。这时就可以使用 elif 语句。具体程序如程序 3.2 所示。

程序 3.2　根据成绩完成详细的成绩评定：

```
1:  mark = int(input("Enter the mark : "))
2:  if mark < 60:
3:      print("The grade is fail.")
4:  elif mark < 75:
5:      print("The grade is pass.")
6:  elif mark < 85:
7:      print("The grade is good.")
8:  elif mark <=100:
9:      print("The grade is excellent.")
10: else:
11:     print("The mark is invalid.")
```

输出：

```
Enter the mark : 90
The grade is excellent.
```

分析：

程序的第 1 行和程序 3.1 一样，剩下部分展现了 if…elif…else 语句的实现。程序流程图如图 3.3 所示。

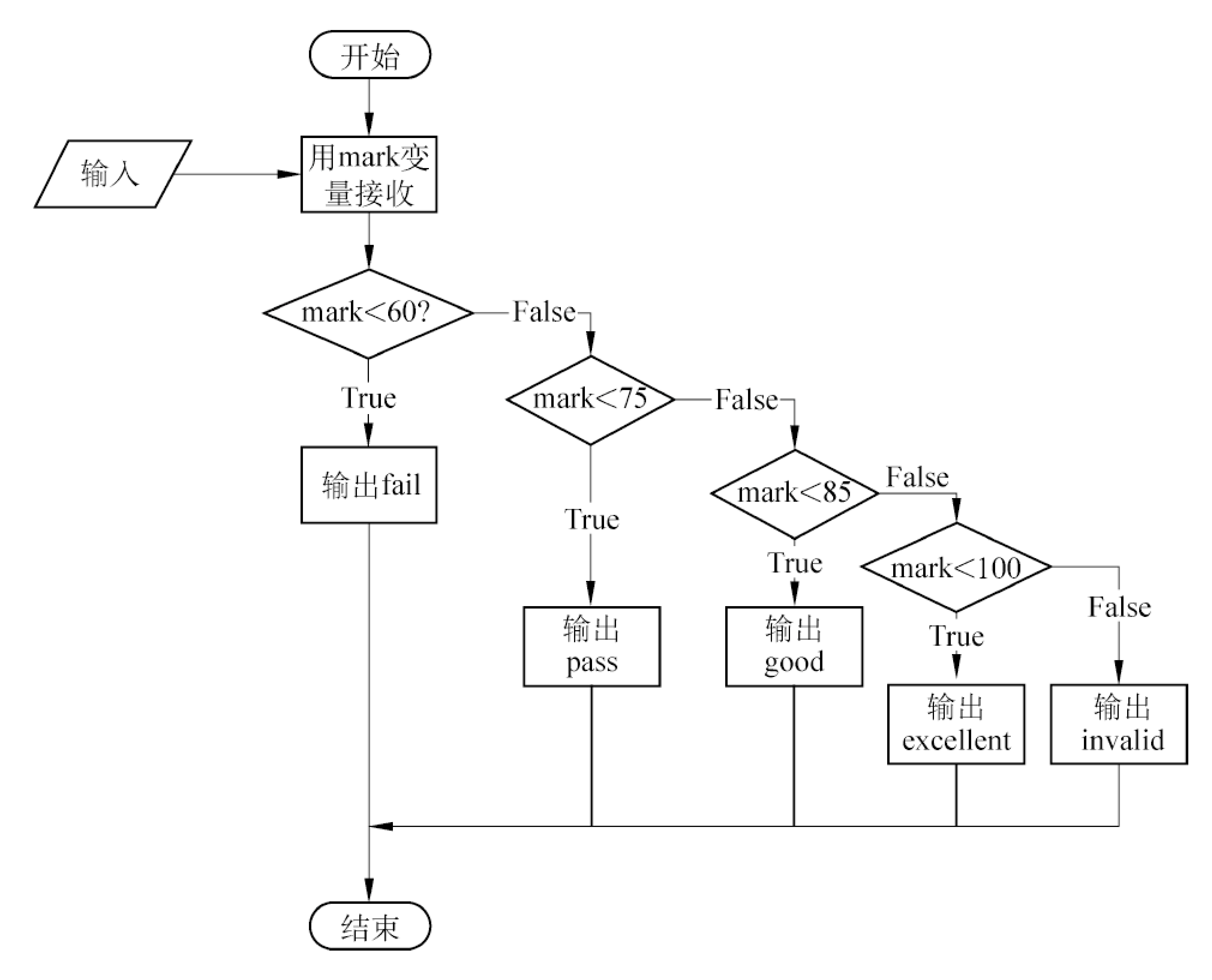


图 3.3　程序的流程图

注意：

如果你之前接触过 C/C++，那么很遗憾，Python 中没有 switch 语句，但是你可以使用 if…elif…else 语句来做同样的事情。

通过上面的例子，我们已经了解了 if…elif…else 语句的结构，接下来看一个模拟机器检测 PM2.5 值的例子。在机器内部，通过传感器检测 PM2.5 的值（见图 3.4），然后再判断这个值对应的空气情况。具体程序如程序 3.3 所示。

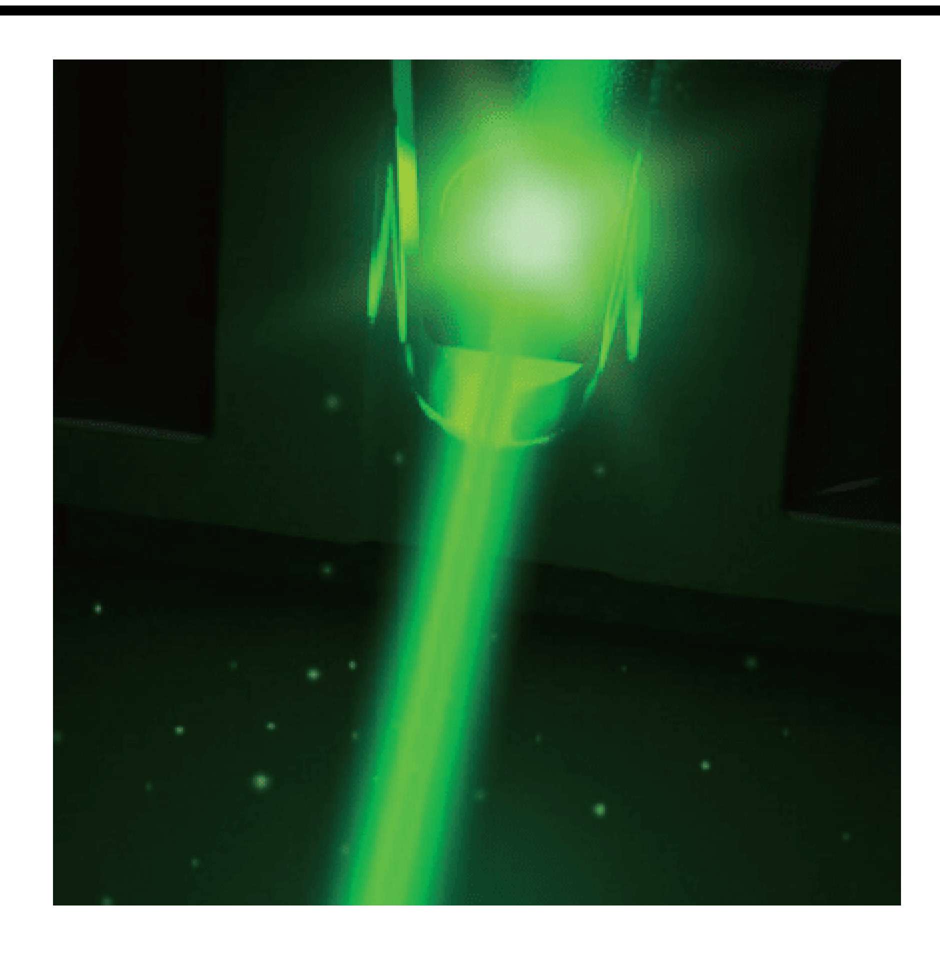


图 3.4　正在工作的传感器

程序 3.3　模拟机器检测 PM2.5：

```
# suppose that the PM2.5 value has been detected to be 80
pm_val = 0b1010000 # 80

if pm_val > 0 and pm_val < 0b100011: # 0 - 35
    print("The air quality is excellect.")
elif pm_val < 0b1001011: # 35 - 75
    print("The air quality is good.")
else:
    print("The air quality is polluted.")
```

输出：

```
The air quality is polluted.
```

分析：

程序使用了一个 if…elif…else 语句模拟了机器检测 PM2.5 的例子，由于我们已经学习了条件控制语句，所以理解这个例子并不难。但是要注意的是，真正在机器中使用的是二进制，在这里先简单介绍一下关于进制的内容。

在 Python 中，我们分别使用 0b、0o、0x 作为二进制数、八进制数以及十六进制数的开头，后跟具体要表示的数，就像程序 3.3 中二进制数 1010000 需要写成 0b1010000。另外，十进制数（假设存入变量 num）也可以使用 bin(num)、oct(num)、hex(num)分别生成 num

中十进制数对应的二进制数、八进制数、十六进制数。

注意：

使用 bin、oct、hex 这 3 个函数的返回值是字符串类型。

3.2　循环控制语句

本节将详细介绍 Python 中的 while 循环语句和 for 循环语句以及一些控制语法。

3.2.1　理解 Python 中的循环语句

Python 提供了 while 循环语句和 for 循环语句以及 break、continue 等循环控制语句，循环语句执行的一般流程如图 3.5 所示。

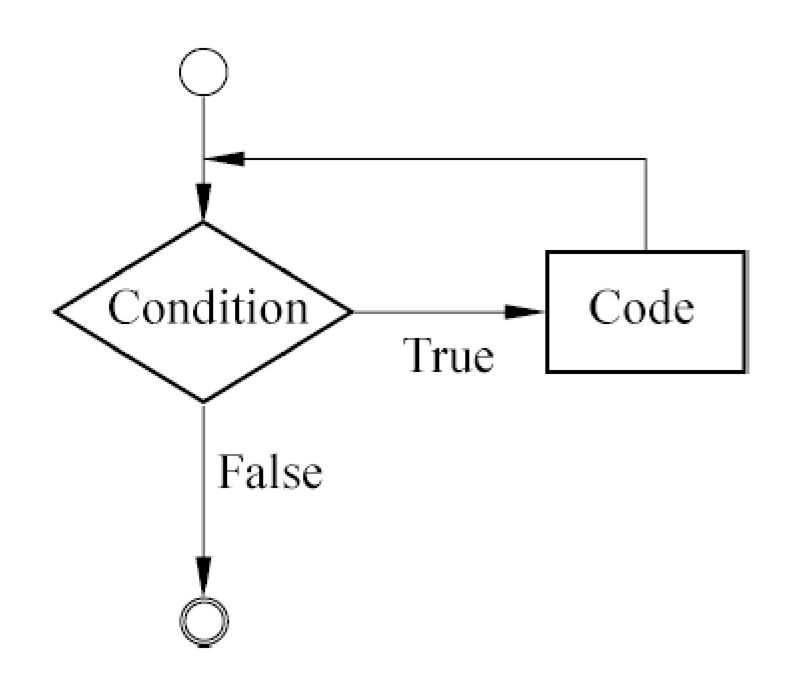


图 3.5　循环语句流程图

循环语句允许执行一条语句或语句组多次，允许更复杂的执行路径。下面来学习本章最重要的两个语句：while 循环语句和 for 循环语句。

3.2.2　while 循环语句

具体语法：

```
while expression:
        suite_to_repeat
```

while 循环语句中的 suite_to_repeat 会一直循环执行，直到表达式（expression）的值为假（False）。执行流程图如图 3.6 所示。

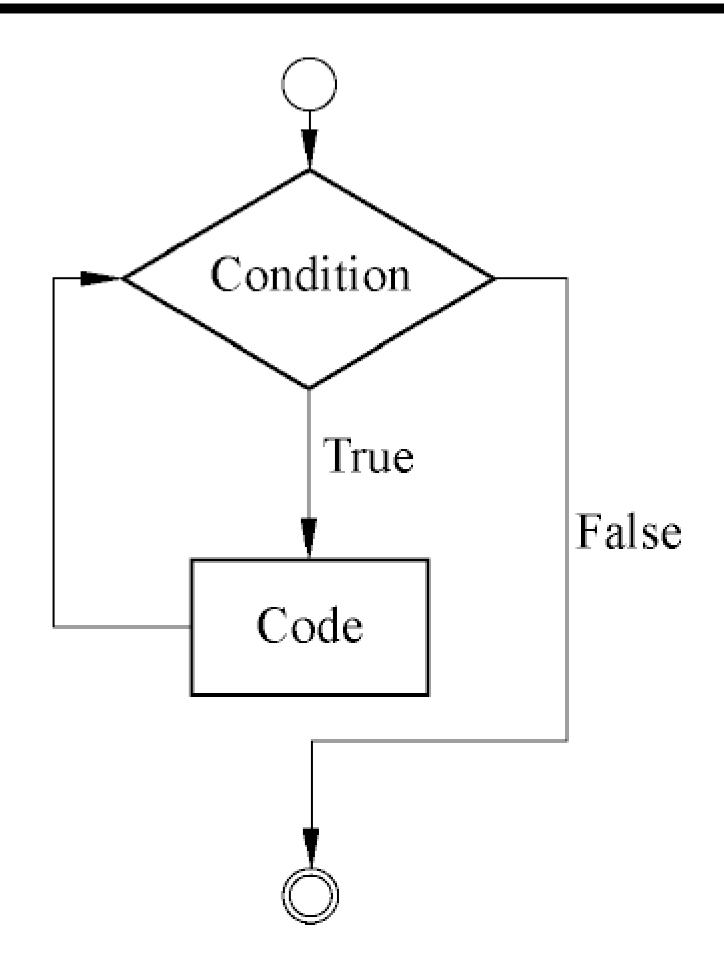


图 3.6　while 循环语句流程图

接下来用一个日常的例子来更加明了地展示 while 的用法：假如老师让你完成全班同学的成绩评定（全班同学的排列是有序的），你该怎么做呢？在这种情况下，需要为全班的每一个同学都重复执行一次评定代码，也就是在循环中执行评定代码。因为已经将全班同学的排列设置为有序的，这使得我们不用考虑班级成员的姓名，直接用数字代替。具体程序如程序 3.4 所示。

程序 3.4　全体成绩评定：

```
num = int(input("Enter the class size : "))

count = 0
while count < num:
    mark = int(input("Enter the mark : "))
    if mark < 60:
        print("The No.{0} grade is fail.".format(count+1))
    elif mark < 75:
        print("The No.{0} grade is pass.".format(count+1))
    elif mark < 85:
        print("The No.{0} grade is good.".format(count+1))
    else:
        print("The No.{0} grade is excellent.".format(count+1))

    count += 1
```

输出：

```
Enter the class size : 2
Enter the mark : 22
The No.1 grade is fail.
Enter the mark : 64
The No.2 grade is pass.
```

分析：

程序的第 5～15 行是循环执行部分，从缩进格式可以看出它们是在 while 循环中的，注意最后一行的 count += 1，这是 count = count + 1 的简写。我们设定 count 让它在每次的循环之后加 1，以此达到控制程序的目的。直到 count 自增到不再小于 num 的时候循环结束，也就是说，当 count 与 num 相等时不会进入循环结构中，因为这个特性而且我们还要保证循环的次数，所以将 count 的初值设为 0，循环条件设置为 count<num，但这样又与编号差 1，所以在打印时我们使用 count+1（也可以让 count 初值为 1，循环条件为 count<=num，这样在打印的时候用 count 即可）。

注意：

（1）在循环中使用自增变量（就是程序 3.4 中的变量 count）控制循环条件是一种很常见的用法。

（2）对于 while 循环的循环控制变量要小心对待，因为有些值永远不会为 False，这样的循环是永远不会结束的，当然这也不一定是坏事，这取决于实际情况。

通过上面的例子，while 语句的用法已经明确了，接下来我们用一个等差数列的例子进一步讨论 while 的用法。这是一个简单输出指定项数的等差数列（首项为 2/3，公差为 2/5）各项的程序，具体的各项为 2/3、16/15、22/15、28/15（这里只列举数列的前 4 项）。我们将在这个例子中引入 Python 中分数的表达方法，这样在本例之外可以使用 Python 做更多的事情。关于程序的具体解析在分析中有更详细的说明。具体程序如程序 3.5 所示。

程序 3.5　输出指定项数的等差数列的各项：

```
1:    from fractions import Fraction
2:
3:    size = int(input("Please input the size of sequence : "))
4:    count = 0
5:    step = 0
6:    while count < size:
7:        print(Fraction(2, 3) + step)
8:        step += Fraction(2, 5)
9:        count += 1
```

输出：

```
Please input the size of sequence :6
2/3
16/15
22/15
28/15
34/15
8/3
```

分析：

在程序 3.5 中，我们使用 Python 提供的 Fraction 方法来表示分数，它的原型是：

```
Fraction(numerator, denominator)
```

也就是说，它接收两个参数，第一个作为分子，第二个作为分母。它可以直接与数运算（如 Fraction(5,16)+3），也可以在 Fraction 对象之间相互运算〔如 Fraction(1，16)*Fraction(3,16)〕。另外，关于模块、对象的详细使用方法后文会介绍，在这里你可以简单地将 Fraction(2, 3)理解为我们平时笔算时写的 2 / 3。

现在继续回到程序中，第 1 行中的语句可以简单地理解为将 Fraction 模块导入到此程序中，这样就能用 Fraction 方法了。接下来我们使用 step 表示公差，在 while 循环中不断地加到首项上，以此构造一个等差数列，并在最后输出。while 循环还是和上一个例子一样，是通过自增变量控制的。程序流程图如图 3.7 所示。

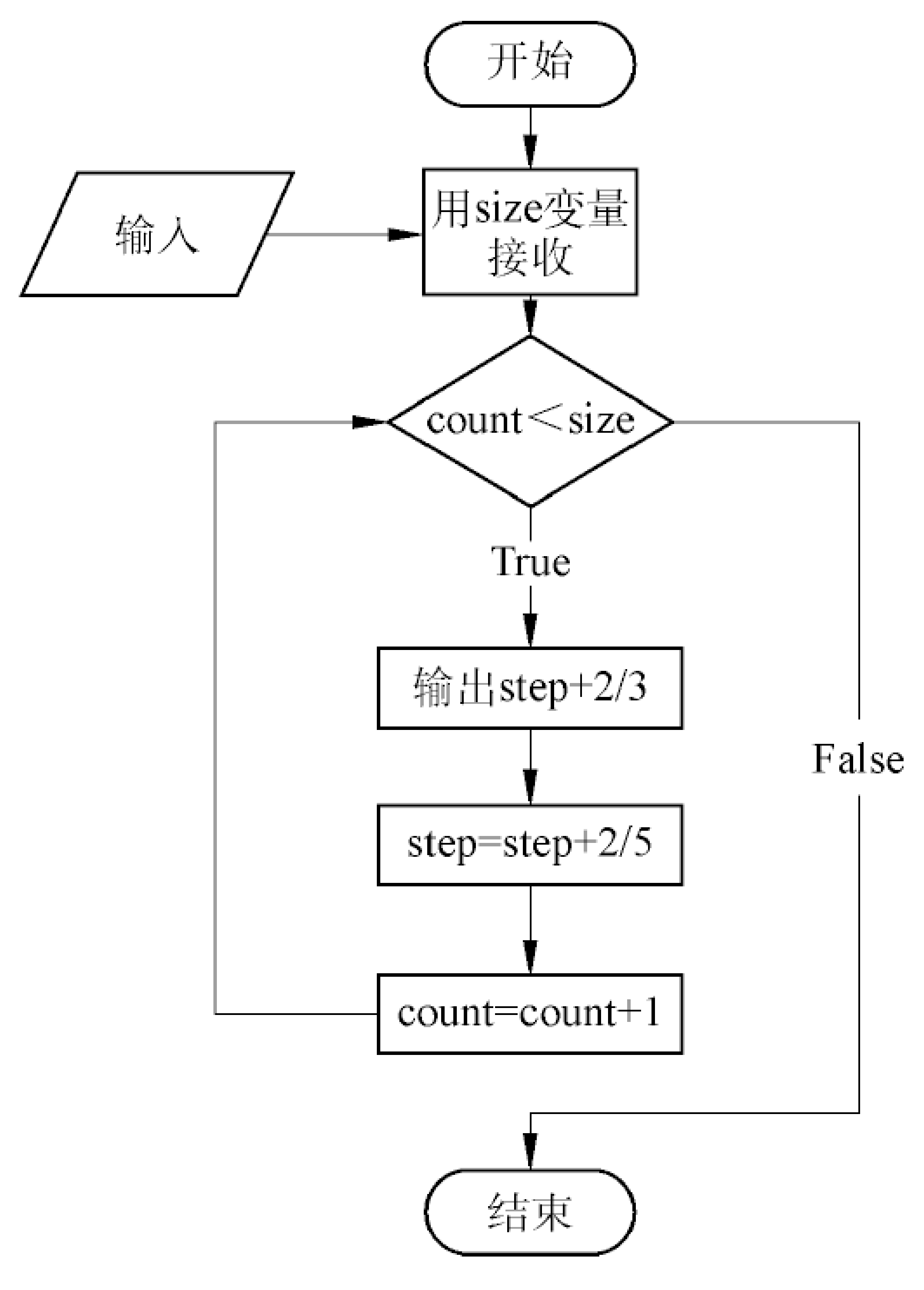


图 3.7　程序流程图

思考题：

Fibonacci 数列是数学中一个非常重要的数列，它指数列前两项为 1 并且从第 3 项开始往后，每一项都等于前两项之和。例如，数列的前 5 项为“1, 1, 2, 3, 5”。在这里我们来思考一下，如何使用 while 循环语句输出最后一项小于 1000 的 Fibonacci 数列。

关于 Fibonacci 数列还有一种递归实现，关于递归的概念以及具体实现方式在本书的函数部分会详细讲解。

3.2.3　for 循环语句

具体语法：

```
for iter_var in iterable:
        statement(s)
```

for 循环是 Python 提供的另一个循环机制，它会依次访问一个已选定的可迭代对象（例如，我们很常见的字符串、序列或是元组）中的所用元素，即遍历可迭代对象，这个过程在具体语法中就是将 iter_var 设置为 iterable 中的当前元素，当 iterable 中的所有的条目（如序列中的项）都处理过后结束循环。执行流程如图 3.8 所示。

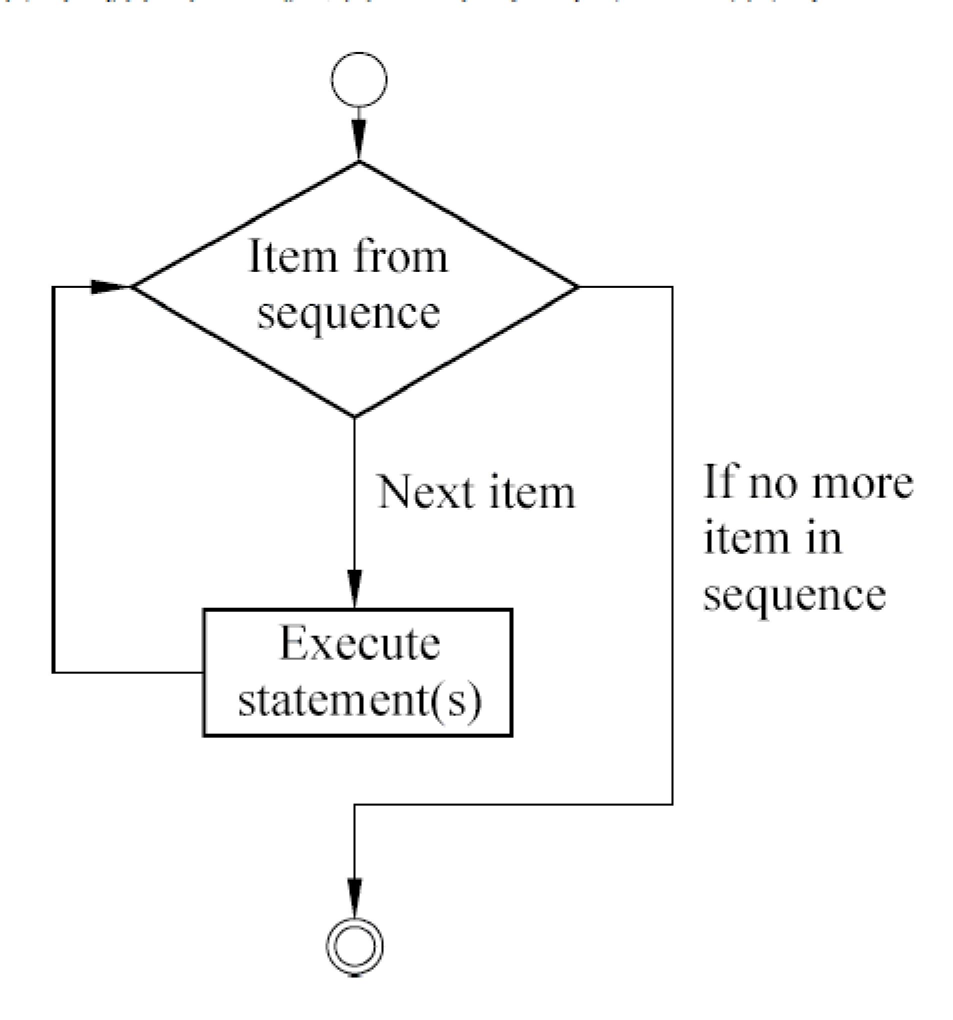


图 3.8　for 循环语句流程图

注意：

读完上面这一段 for 循环的具体介绍，你可能会有疑问，什么是迭代？你可以将迭代理解为一种环形结构，它也叫循环，while 和 for 语句也被称为迭代语句。

如果你之前接触过 C/C++或者 Java 的话，请一定注意，Python 中的 for 语句与其他编程语言的用法不同。

到这里我们已经知道了 for 循环会遍历一个可迭代对象，下面来看看如何使用它。我们介绍两种基本方法：使用序列项或者序列索引。假如已经拿到你们小组的名单，而老师需要你将小组成员的名字和组号一同输出。诸如此类的情况，for 语句可以很好地实现。

程序 3.6 会使用序列项迭代来遍历输出名单上的名字。

程序 3.6　使用序列项迭代：

```
name = ['Sam', 'Lily', 'Tom', 'William', 'Michael', 'Leo']

for item in name:
    print(item, '--Group 5 ')
```

输出：

```
Sam --Group 5
Lily --Group 5
Tom --Group 5
William --Group 5
Michael --Group 5
Leo --Group 5
```

分析：

程序中 name 是一个列表，代表着小组成员名单，我们用 for 语句迭代这个列表，其中 item 变量在每次的循环中都会被赋值为列表中的某个特定的元素，以供循环中的语句使用。

介绍完使用序列项迭代之后，我们来看看另一种方法，使用序列索引迭代。不同于使用序列项迭代，使用序列索引迭代就如同字面意思所说，操作的重点在于拿到序列索引号并通过它获取序列中的项。还是上面的例子，我们改用序列索引迭代实现，具体程序如程序 3.7 所示。

程序 3.7　使用序列索引迭代：

```
name = ['Sam', 'Lily', 'Tom', 'William', 'Michael', 'Leo']

for itemIndex in range(len(name)):
    print(name[itemIndex], '--Group 5 ')
```

输出：

```
Sam --Group 5
Lily --Group 5
Tom --Group 5
William --Group 5
Michael --Group 5
Leo --Group 5
```

分析：

本程序和程序 3.6 看上去相似，但是并不相同，在本程序中你会发现一个略显生疏的函数——range，它的作用是返回一个序列，这个序列正好就是我们需要的迭代序列。我们将 range 的参数设置为姓名列表（name）的长度，这样从 range(len(name))中获取的数即可作为 name 的索引数(注意使用 range 函数获取序列的值是从 0 开始，len(name)-1 结束），再使用 name 列表的切片操作来获取与索引值相对应的项，并在 for 循环中完成输出。

看完分析你可能仍对 range 函数存有疑惑，下面我们再详细地谈谈 range 函数。先来看看完整的 range 函数：

```
range(start, end, step)
```

range 函数会返回一个包含所有项的列表，这里我们假设项为 i，i 会满足 start<= i < end，从 start 开始，i 每次递增 step（注意：step 一定不能为 0）。例如，range(2, 6, 2)会返回列表[2, 4]。

我们经常用到的是它的两种简化形式：

```
range(end)
range(start, end)
```

第一种只接收一个 end 值，它的 start 默认为 0，step 为 1。例如，range(3)会返回列表[0, 1, 2]。第二种接收两个值，它的 step 默认为 1。例如，range(2, 4)会返回列表[2, 3]。

当 for 循环涉及数字操作的时候，使用 range 函数是很有用的。

提示：

如果你的程序需要考虑到性能的问题，那么对于上文所讲的序列项迭代和索引迭代，序列项迭代的速度会更快。

注意：

这里，我们已经知道 for 循环会依次访问一个已选定的可迭代对象的所有元素。你是否会疑惑，它是怎样做到的呢？实际上，在 for 循环的内部，它会自动帮我们调用迭代器的 next()方法来达到遍历的效果，它也会捕获 StopIteration 异常使循环结束。这样便可以完

成上文所提到的功能。

现在你应该对于 for 循环有了更深的理解，接下来用一个大家都很熟悉的例子巩固一下。在物理课上你应该做过用打点计时器测速度的实验（人教版高中物理必修 1 第一章），现在假设我们拿到的纸带如图 3.9 所示。

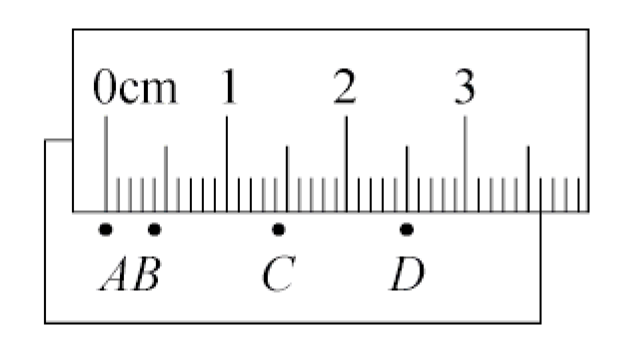


图 3.9　用于实验的纸带

已知打点计时器所用电源频率为 50Hz，AB 距离为 0.0040m，BC 距离为 0.0102m，CD 距离为 0.0106m。我们要求每一段的平均速度。具体程序如程序 3.8 所示。

程序 3.8　求纸带上每段距离的平均速度：

```
item_len = [0.0040, 0.0102, 0.0106]
sec = 0.02
print("The speed is : ")
for item in item_len:
    print("{0} m/s".format( item/sec ), end="    ")
```

输出：

```
The speed is :
0.2 m/s    0.51 m/s    0.53 m/s
```

分析：

本程序涉及物理课中的一个很简单的测速实验，在这里我们假设已经测量完纸带的间距并将它们存储在 item_len 列表中，用 item 遍历获取 item_len 列表中的各段距离值用于 for 循环中的具体操作。由打点计时器所用的电源频率为 50Hz 可知，纸带上两点间隔的时间为 0.02s，得到这些条件后，用 $v = \Delta x/\Delta t$ 来求得每段的平均速度。程序流程图如图 3.10 所示。

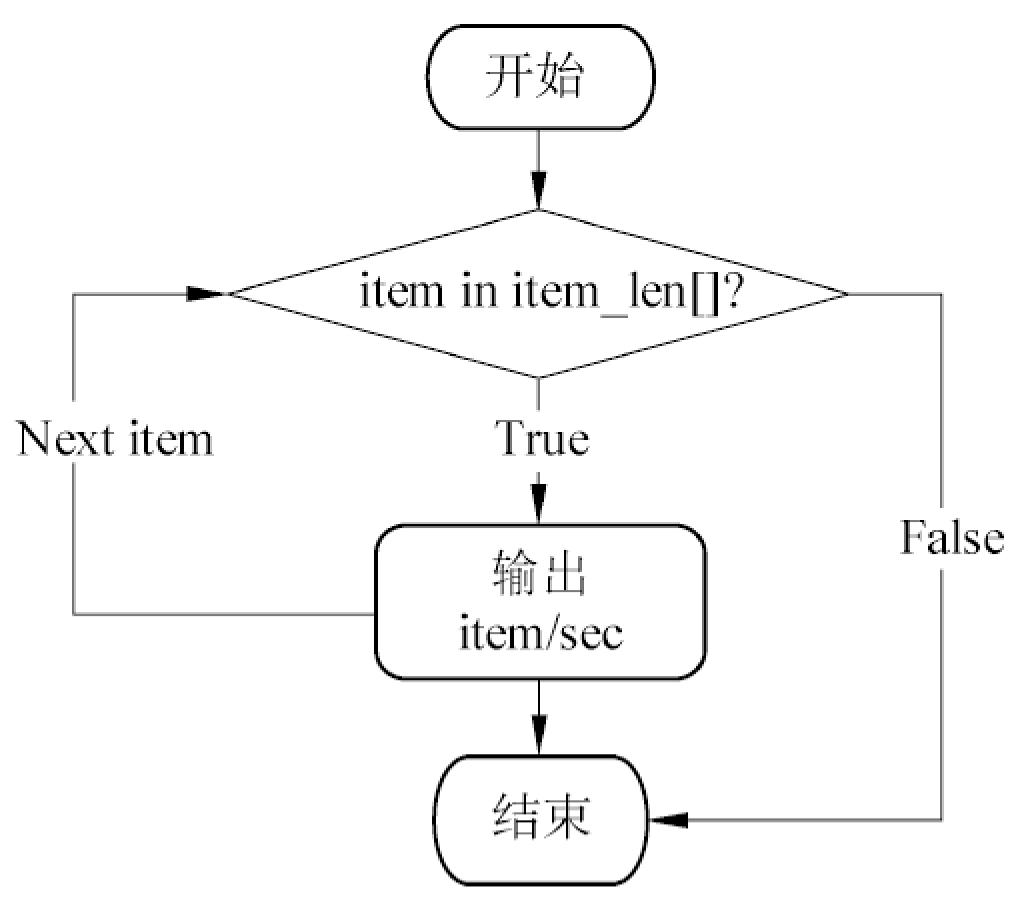


图 3.10　程序流程图

注意：

在编程的时候，我们可以让两个字符串相加达到组成一个字符串的目的，但是，在程序 3.8 中的 print 语句中你不能使用 print("{0} m/s" + item/sec)的形式进行输出，因为此时 item/sec 是 float 类型，这样做会让你的程序报错：TypeError: must be str, not float。像这种从其他信息中构造字符串正是 format()方法大有用武之地的地方。

3.2.4　break 语句

break 语句在控制中很常见，它可以终止当前的循环，即跳出循环语句执行程序中的下一条语句。执行流程图如图 3.11 所示。

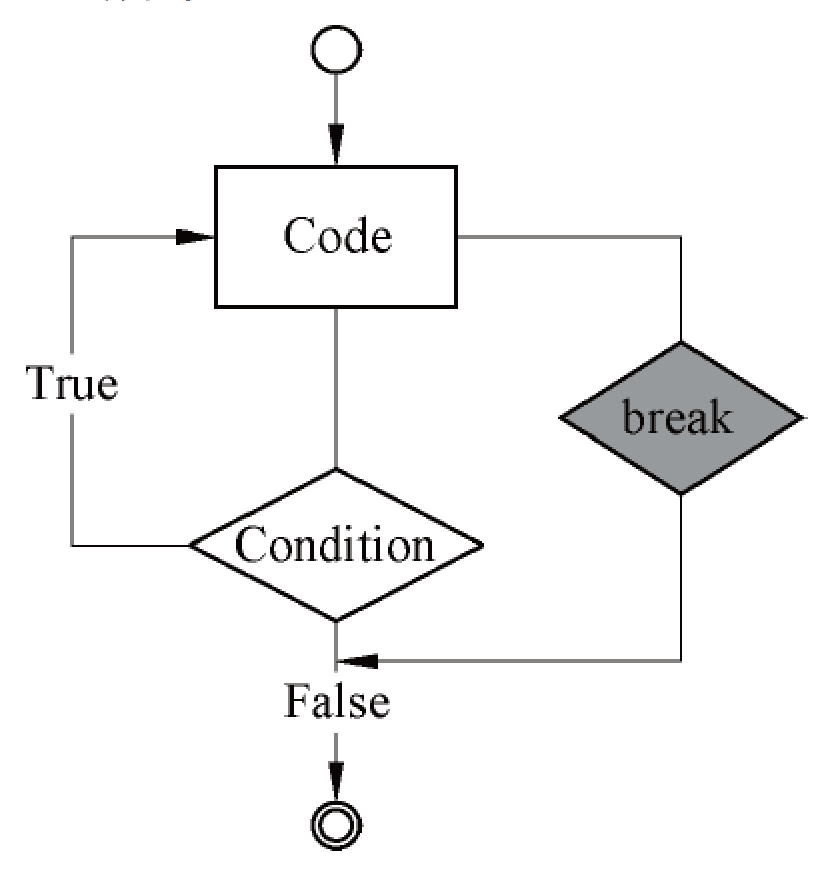


图 3.11　break 语句流程图

现在我们已经对 break 语句有了一定的了解，接下来用一个例子来实践一下。我们都笔算过最大约数，接下来就用 Python 实现算出最大约数。

程序 3.9　求一个数的最大约数：

```
num = int(input('Please enter a number : '))
count = int(num/2)

while count > 0:
    if num % count == 0:
        print('The largest factor of {0} is {1}'.format(num, count))
        break
    count -= 1
```

输出：

```
Please enter a number : 21
The largest factor of 21 is 7
```

分析：

在程序 3.9 中，因为 count 的值是不断递减的，所以第一个能整除 num 的数便是我们要找的，后续的数就没有必要考虑了，使用 break 语句跳出循环。程序流程图如图 3.12 所示。

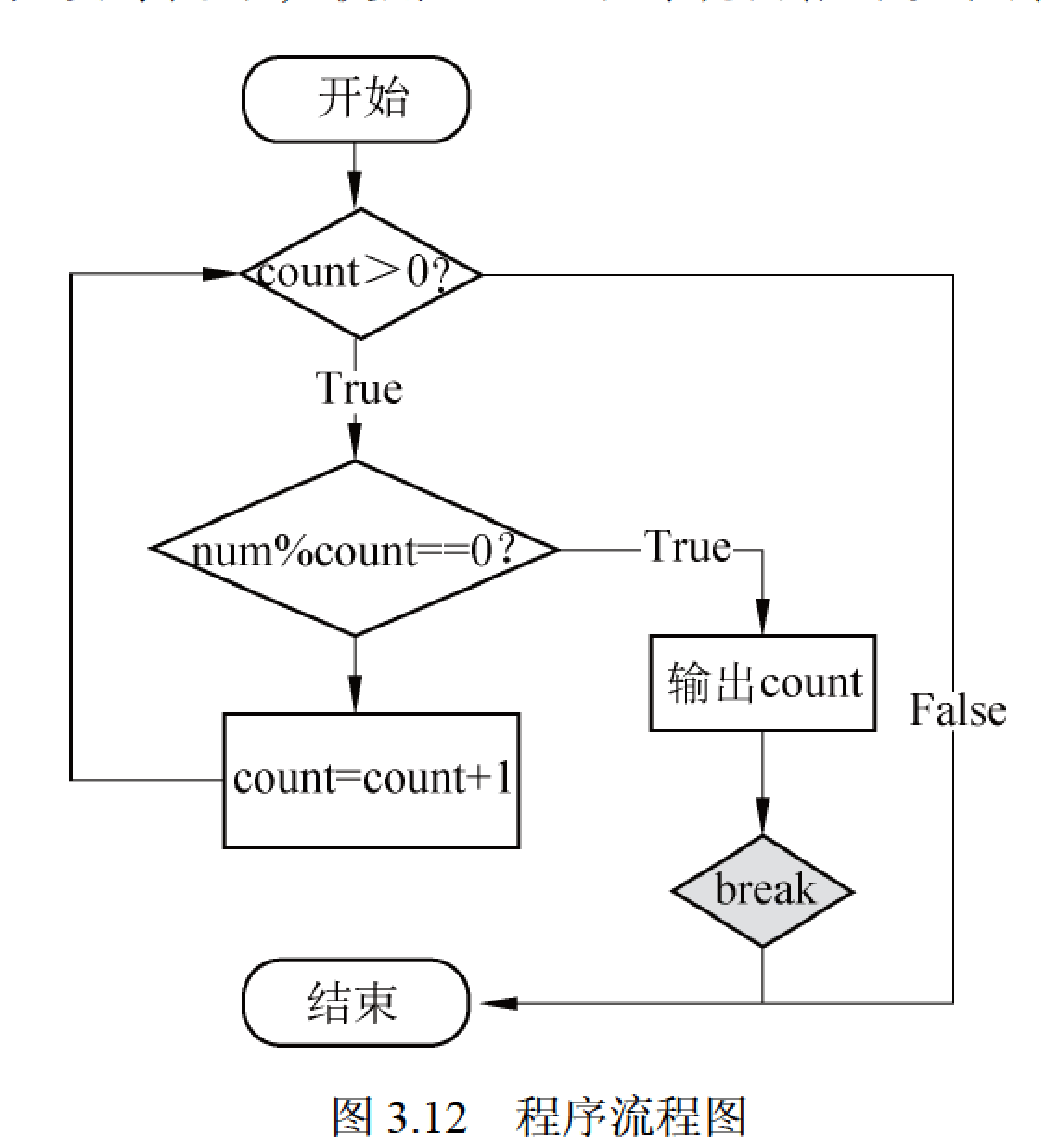


图 3.12　程序流程图

3.2.5　continue 语句

我们可以将 continue 语句理解为跳过当前的循环块，继续该循环的下一次迭代。先来看看 continue 语句的执行流程图，如图 3.13 所示。

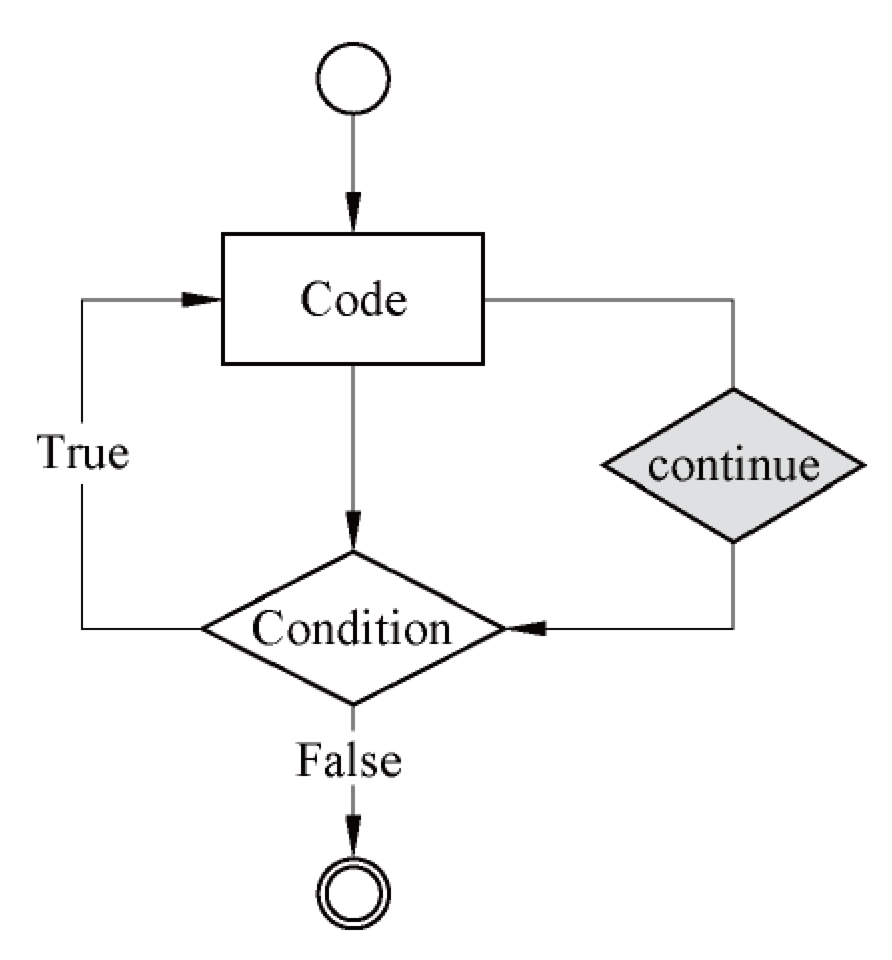


图 3.13　continue 语句流程图

接下来用一个判定提交的选项是否正确（假设正确的选项为 C）的程序来看看 continue 语句是如何工作的。

程序 3.10　判断提交选项：

```
while True:
    write = input("Please write your answer(Options are limited to A/B/C):")
    if write == 'A':
        print("wrong")
        continue
    if write == 'B':
        print("wrong")
        continue
    if write == 'C':
        print("right")
        break
print('Congratulations!')
```

输出：

```
Please write your answer(Options are limited to A/B/C):A
wrong
```

```
Please write your answer(Options are limited to A/B/C):B
wrong
Please write your answer(Options are limited to A/B/C):C
right
Congratulations!
```

分析：

在这个程序中不仅用到了 continue 语句，还用到了许多之前提到过的语句。在第 1 行中使用 True 作为 while 的循环条件以达到可以不断地对选项进行判断的目的，若输入不是 C，那么将使用 continue 语句跳过本次循环，重复下一次循环，直到输入 C 并执行 break 语句，才终止循环并获得 Congratulations!。程序流程图如图 3.14 所示。

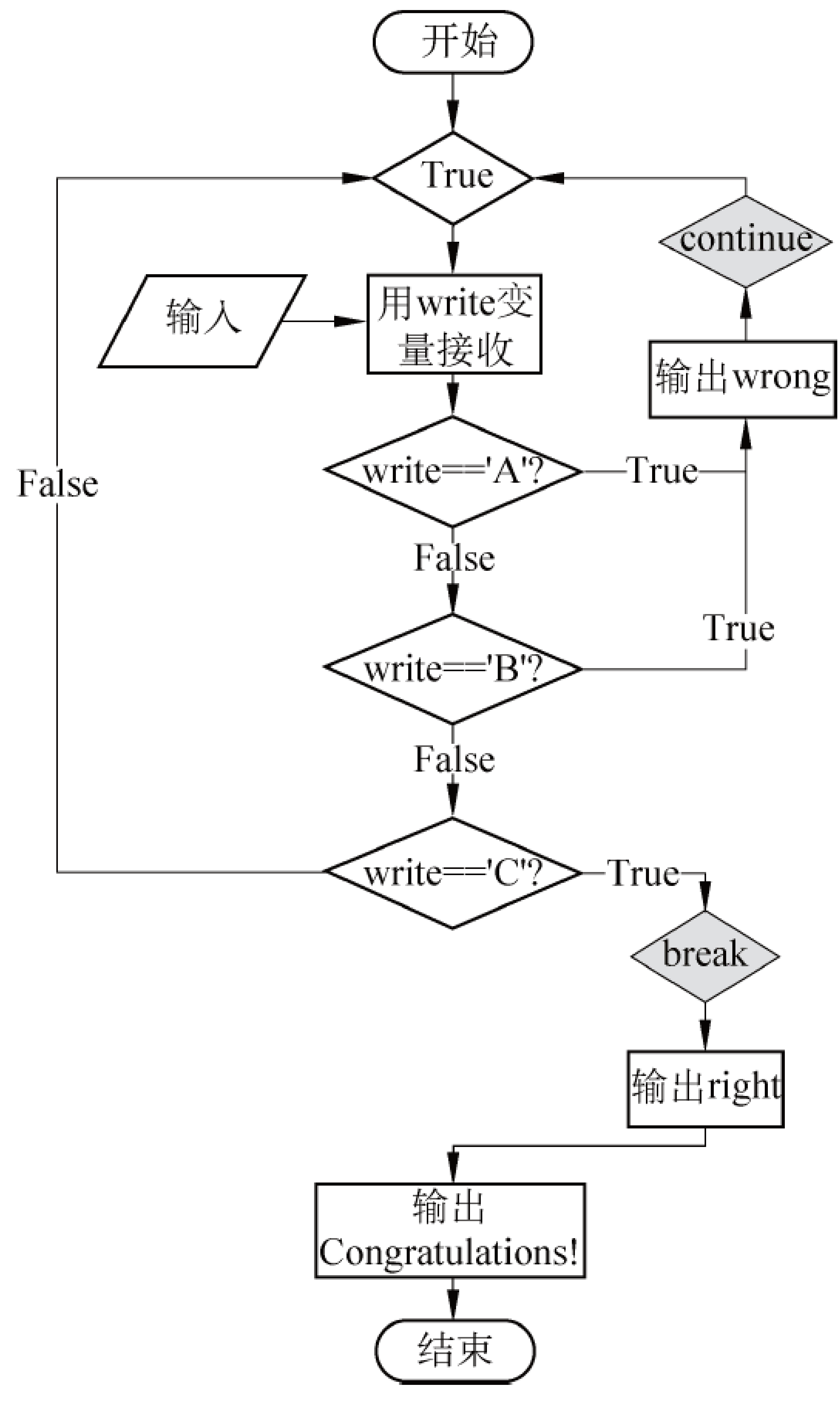


图 3.14　程序流程图

3.3　案例：百钱买百鸡问题

百钱买百鸡是我国古代数学家张丘建提出的一个数学问题，它的内容是鸡翁（公鸡）一值钱五，鸡母（母鸡）一值钱三，鸡雏三值钱一（一钱三只）。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？

在本节中我们用 Python 编写程序对问题求解，并且深入探讨该问题的深层含义。具体的 Python 程序如程序 3.11 所示。

程序 3.11　百钱买百鸡：

```
x_max = int(100/5)
y_max = int(100/3)
for x in range(x_max):
    for y in range(y_max):
        z = 100 - x - y
        if (z%3 == 0) and (5*x + 3*y + z/3 == 100):
            print('公鸡{0}只，母鸡{1}只，小鸡{2}只'.format(x, y, z))
```

输出：

```
公鸡 0 只，母鸡 25 只，小鸡 75 只
公鸡 4 只，母鸡 18 只，小鸡 78 只
公鸡 8 只，母鸡 11 只，小鸡 81 只
公鸡 12 只，母鸡 4 只，小鸡 84 只
```

分析：

程序采用枚举法，即逐个考察所有可能的买法，最后留下符合题意的买法。具体来说，我们使用程序中的双重 for 循环来实现枚举法。首先固定 x 值（公鸡个数），再固定 y 值（母鸡个数），使用 x 和 y 的值依照题意得出 z 的值（雏鸡个数）并判断当前 x、y、z 值是否符合百钱买百鸡，符合输出，不符合则更换 y 值再次求 z 值并判断，直到当前 x 值下的所有 y、z 值全都不符合时，更换 x 值再次进入循环，直到所有 x 值都尝试过后，退出。因为买鸡都是整只的，因此在程序中只需考虑整数情况。程序流程图如图 3.15 所示。

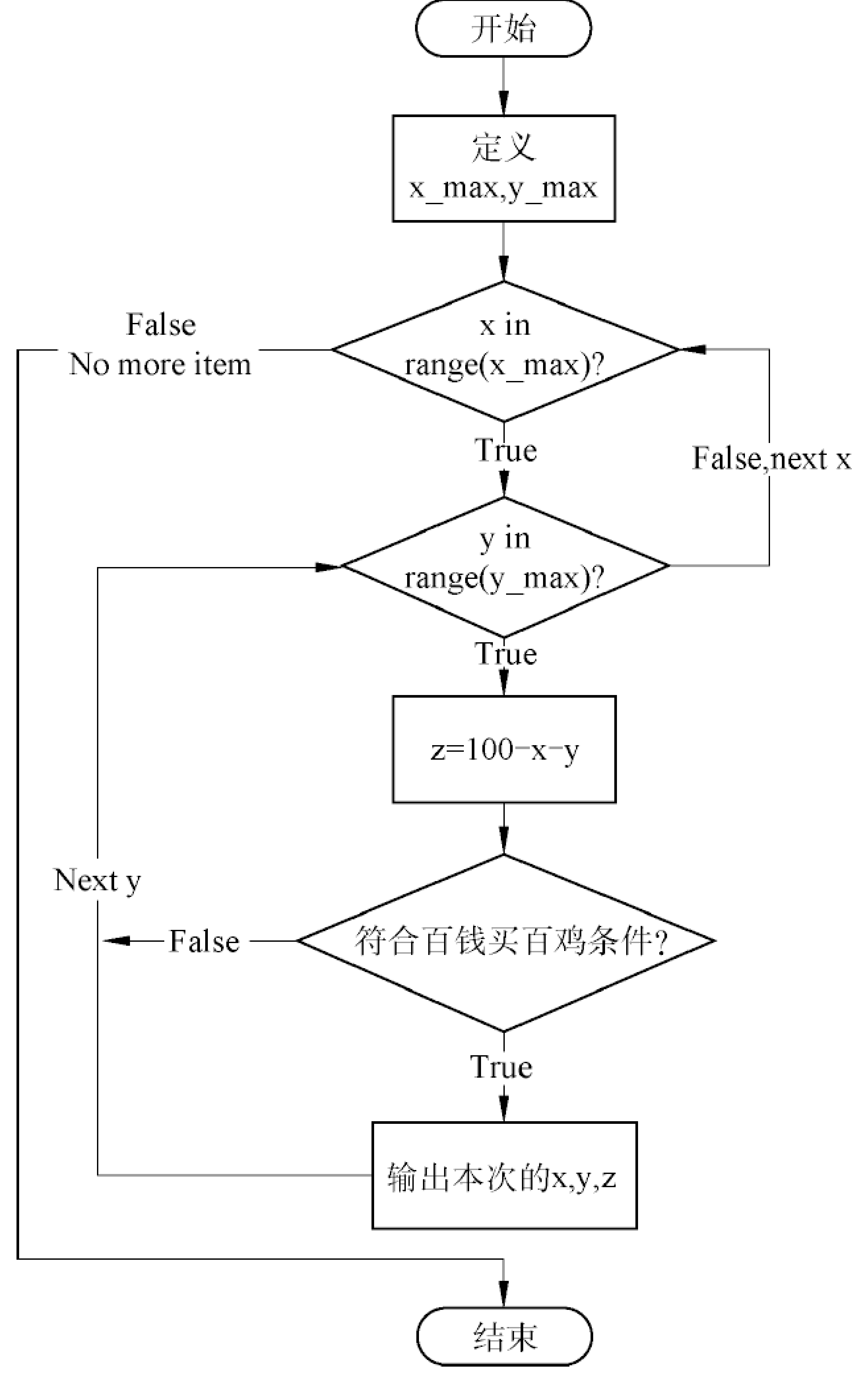


图 3.15　程序流程图

建立数学模型

关于百钱买百鸡的问题并不复杂，我们在解释的时候使用 x 代表要买的公鸡个数，使用 y 代表要买的母鸡个数，z 代表要买的雏鸡个数，完成变量的设置之后根据本题的题意建立了数学模型 $5x+3y+\frac{1}{3}z=100$，其中 z 值为 $100-x-y$，有了这两个式子之后我们就可以对抽象的

题目进行定量分析。在实际应用中，问题可能很复杂，其对应的数学模型也会十分复杂，对此我们可能会用到函数的相关知识来对其进行数学分析，这也是很常用的手段。

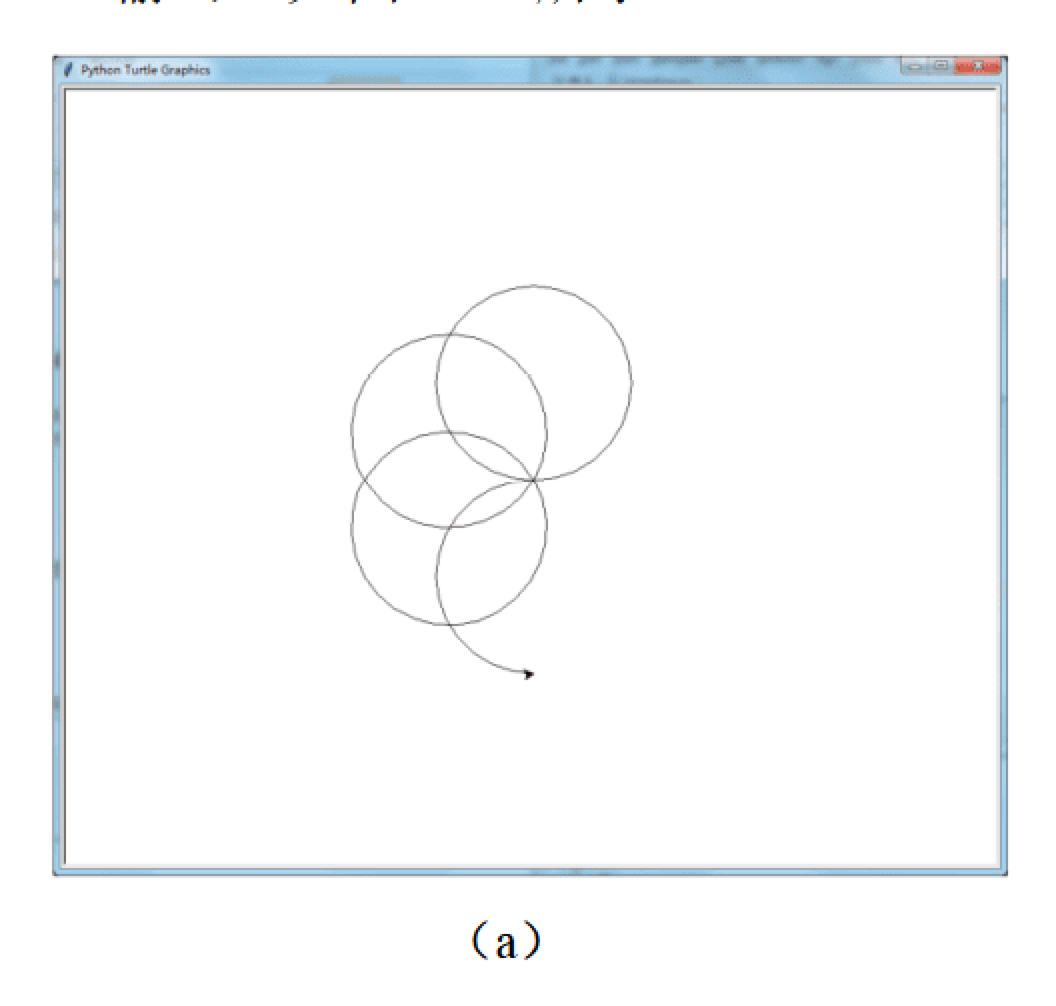
3.4 趣 味 练 习

本节的趣味练习引入一个 Python 中的绘图模块，它会创造一个画布并附有画笔，允许我们对画笔编程，先来看程序 3.12。

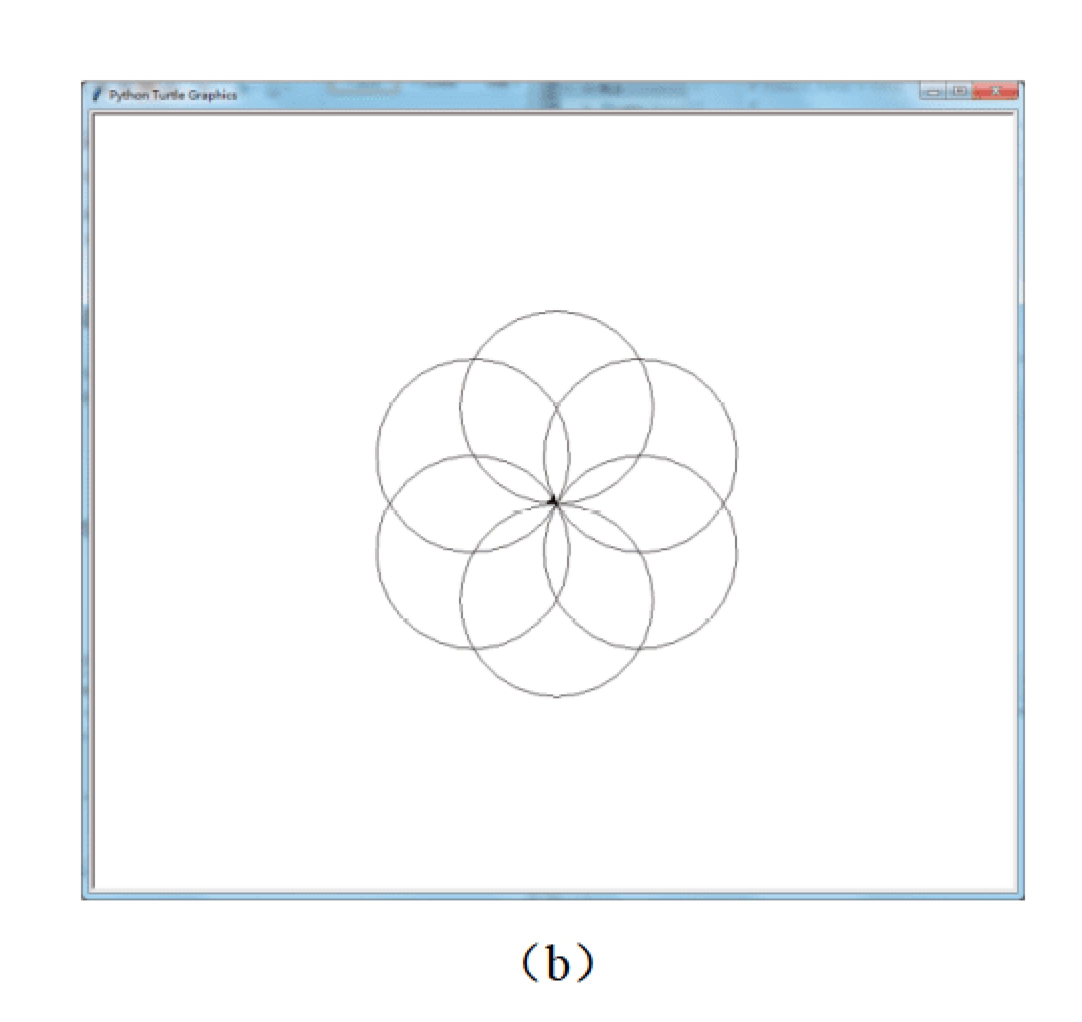
程序 3.12：

```
import turtle
t = turtle.Pen()
for x in range(6):
    t.circle(100)
    t.left(60)
```

输出，如图 3.16 所示：



（a）



（b）

图 3.16　程序 3.12 输出结果

点睛：

首先我们将模块 turtle 引入，它是这个画图程序用到的模块，接着我们将变量 t 定义为画笔，在 for 循环的迭代过程中不断地在画布中画圆，可以看到，在循环的迭代中使用 circle() 指定了在画布中画的是一个圆，同时传入参数 100 表示这个圆的半径为 100 个单位，left(60)

表示将画图的起笔逆时针旋转 60°，这个会影响到下次画图位置。

输出图 3.16（a）是绘图的过程，图 3.16（b）是最后完成时的图像。实际运行只会输出一个画图界面。

接下来我们再来看看趣味程序 3.13，从程序的角度来说，相比于程序 3.12 它更为丰满。

程序 3.13：

```
import turtle
answer = input("Do you want to see a spiral?")
if answer == 'y':
    print("Working...")
    t = turtle.Pen()
    t.width(2)
    for x in range(100):
        t.forward(x*2)
        t.left(88)
else:
    print("what a pity!")
    print("Done!")
```

输出，如图 3.17 所示：

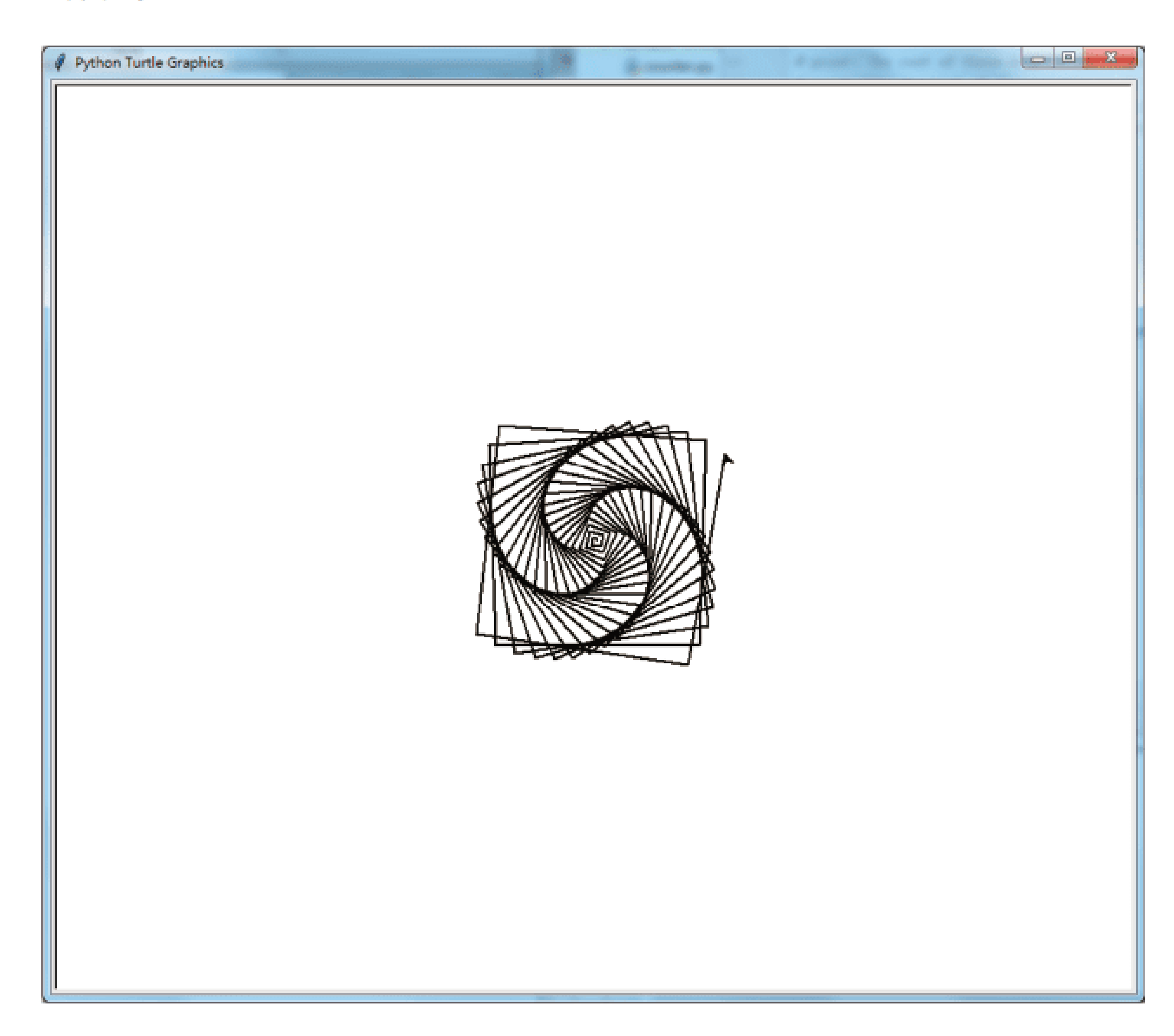


图 3.17　程序 3.13 输出结果

点睛：

有了程序 3.12 的铺垫，理解程序 3.13 并不算难，在这里我们只对设置画笔部分简单分析一下。程序的第 6 行设置了画笔的粗细，第 8 行设置了在每次迭代中要向前画出的长度，第 9 行设置了下次画图时的偏转方向，在画图时可以根据需要更改它的值。

输出图 3.17 中只是截取了画笔在工作时某一瞬间的图像，它是一个动图。

趣味一刻：

如果将程序 3.12 第 4 行改为 t.circle(−100)，程序 3.13 第 8 行改为 t.forward(−x*2)的话，对应的输出图像会有什么效果?

3.5　总　　结

本章学习了条件控制语句和循环控制语句，它们是 Python 中最常用的部分，在本书的其他章中我们还会频繁地看到它们的身影。在实际编程中它们也担当着重要的角色，因此，尽快去习惯使用它们是很有必要的。

在学习完本章之后我们会发现，其实本章提到的所有语句都不是太难。在实际编程中，对于条件控制语句，重点应放在根据实际情况的判断处理好每一种分支。而对于循环控制语句，应侧重于找到需要循环处理的部分以及如何控制整个循环流程。

3.6　练　　习

（1）参考 while 语句和 if 语句的流程图画出程序 3.4 的具体流程图。

（2）输出首项为 3/5，公比为 2/7 的等比数列的前 10 项，并循环计算这个数列的前 10 项和。

（3）完善程序 3.8，让程序计算并输出 B 点的瞬时速度更接近于多少。

披荆斩棘（选做）：

（1）打印如下所示的 1～6 三角阵列。

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4 5 6

（2）对列表 [11,5,9,2,7,1,13,4] 进行冒泡排序。

第 4 章　数 组 操 作

本章主要介绍的是列表（list）、字典（dictionary）和元组（tuple），它们是一种能够将数据聚合（存储）在一起的操作。也可以将它们理解为用来存储一系列相关数据的集合。通过本章的学习，需要掌握：

- ❑ 列表及其操作。
- ❑ 字典及其操作。
- ❑ 元组及其操作。
- ❑ 什么是排序和查找。
- ❑ 冒泡排序算法。
- ❑ 二分查找算法。
- ❑ 理解算法分析。

4.1　列　　表

列表是 Python 中的内置对象，它是 Python 中最灵活的有序集合对象。它有些功能和之前提到的字符串类型很像，但比字符串要灵活得多。列表可以包含任何种类的对象，不管是数字、字符串甚至是另外几个列表。同时，列表和字符串有着本质的不同，它是可变对象，我们可以通过指定偏移值和分片，列表方法调用以及删除语句等方法来实现在原处修改列表的操作。

接下来先通过一个关于分类加法计数（人教版高中数学选修 2-3）的程序来了解一些关于列表的简单操作。这个程序是计算从北京到上海一共有多少种不同的走法（见图 4.1）。

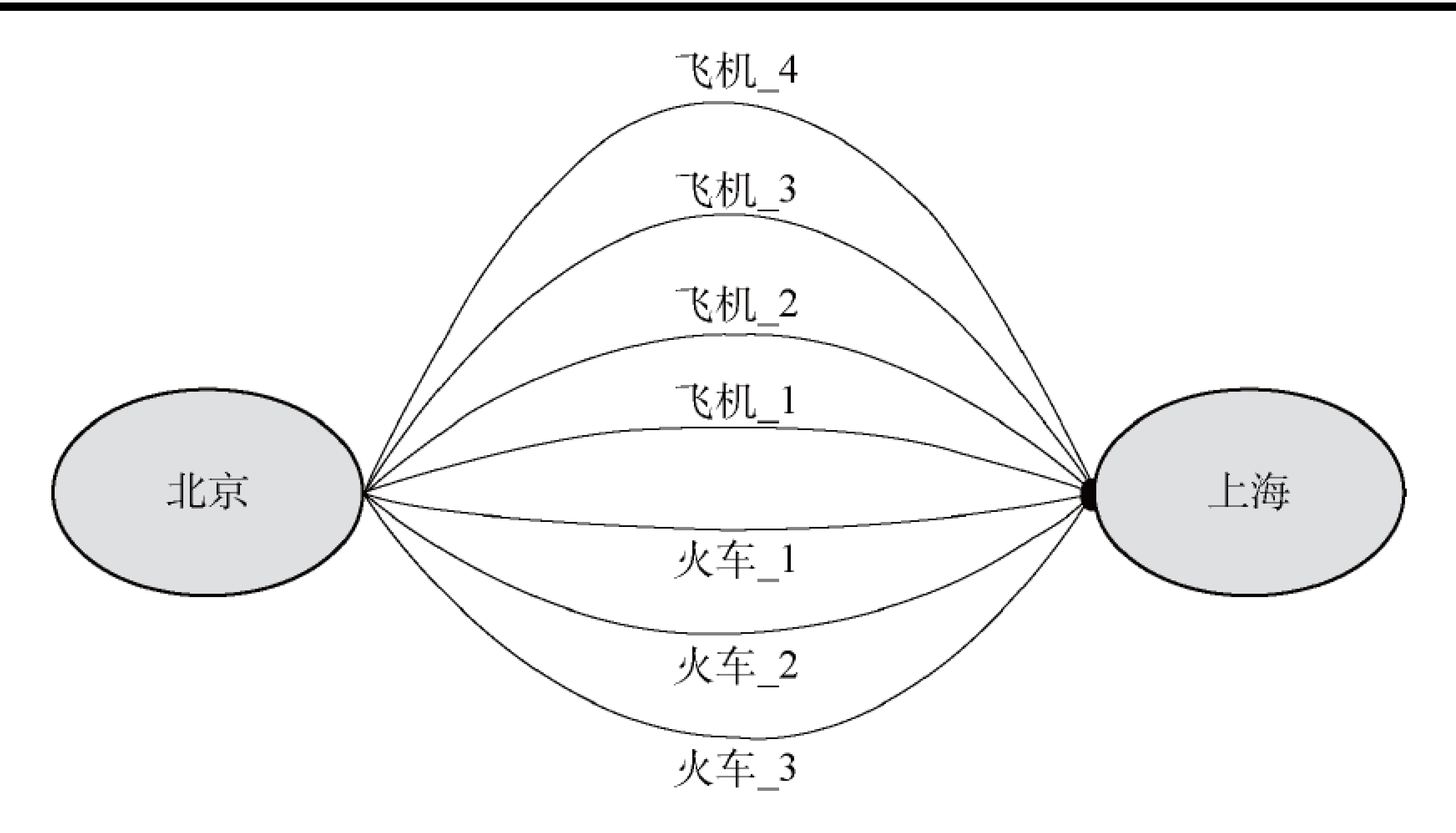


图 4.1　分类加法图解

程序分为两段，第一段将不同的走法存入一个列表中，第二段将所有走法都展示给用户（将项从列表中取出）。具体程序如程序 4.1 所示。

程序 4.1　计算北京到上海的走法：

```
list = []
for i in range(4):
    list.append("Airplane_{0}".format(i+1))

for i in range(3):
    list.append("Train_{0}".format(i+1))

print("The num of the way from Beijing to Shanghai is : {0}".format(len(list)))

for i in list:
    print(i)
```

输出：

```
the num of the way from Beijing to Shanghai is : 7
Airplane_1
Airplane_2
Airplane_3
Airplane_4
Train_1
```

```
Train_2
Train_3
```

分析：

由分类加法计数原理：若完成一件事有两类不同的方案，其中第一类方案中有 m 种不同的方法，第二类方案中有 n 种不同的方法，那么完成这件事一共有 $N=m+n$ 种不同的方法。从北京到上海的走法数即飞机和火车这两类方案的方法总和。

下面回到程序中，由于我们在第 3 行的循环中直接使用了 list 的 append()方法，所以在用之前要声明一下 list 是一个列表，如程序的第 1 行所示。我们使用“[]”表示一个空列表并赋值给 list。声明列表之后，使用刚刚提到的 append()方法并放在循环中。使用两个循环将北京到上海的走法附加到列表 list 中。

append()方法将项插入列表尾，注意列表的索引值同字符串相同，都是从 0 开始的。程序的第 8 行，使用一个 len 函数输出 list 的长度，由上文的分类加法可知，list 的长度就是方法总和即要求的方法数。最后，我们来看第 10 行的 for 循环，因为列表是可循环对象，所以可以直接使用 for…in 语句遍历列表的所有元素，将每项输出。

通过程序 4.1，是不是对列表有了一定了解？接下来我们列出一些关于列表的常用操作，这些都是很有代表性的操作，如表 4.1 所示。

表 4.1　常用的列表操作

序号	操作	解释
1	L = []	定义一个空的列表
2	L = [0, 1, 2]	定义一个项为数字 0,1,2 的列表
3	L = ['abc', ['def', 'ghi']]	定义一个项为字符串、列表的列表。注意列表中含有列表的情况，不管列表中含有什么，它们都是当前列表的项
4	L = list('list')	使用 list 函数定义列表，同语句 L = ['l', 'i', 's', 't']含义一样
5	L = list(range(-1, 3))	使用 list 函数定义列表，同语句 L = [-1, 0, 1, 2]含义一样
6	L[i]	切片语句（这个操作和字符串的很像），返回列表的第 i 项。注意若是提取像序号 3 操作中的索引为 1 的项，提取出的还是个列表，那么对于这个项，可以用 L[1][1]提取列表中的列表的第 2 项
7	3 in L	3 是否是列表 L 中的项，返回一个布尔值
8	L * 3	返回值为列表内容乘以 3，但还是一个列表
9	len(L)	返回列表的长度
10	L1 + L2	返回将列表 L1 的内容和列表 L2 的内容相加的列表

续表

序号	操作	解释
11	L[i] = X	将列表 L 中索引为 i 的项重新复制为对象 X
12	L.pop()	移除列表 L 中的元素（默认为最后一个）并返回该元素的值。也可以使用 L.pop(index)来删除索引值为 index 的值
13	L.append(X)	将对象 X 添加到列表尾
14	L.extend([10, 22, 44])	将列表[10, 22, 44]一次性追加到列表尾
15	L.insert(i, X)	将对象 X 添加到列表 L 索引值为 i 的地方
16	L.count(X)	返回对象 X 在列表中出现的次数
17	L.remove(X)	移除列表 L 中与对象 X 匹配的第一个项
18	del L[i]	删除列表 L 中索引为 i 的项
19	del L[i:j]	删除列表 L 中[i:j]所指定的项

下面再简单介绍下关于表 4.1 的一些知识点。表中的 L、L1、L2 表示已经定义好的列表，如果要使用列表的操作的话一定要先声明，否则会报错。表中使用 i 作为列表的索引值，一定记住列表的索引值是从 0 开始的，X 表示某一对象，它可以是任何类型的数据。

若是对表中的某些操作还不理解，可以在 PyCharm 中定义一个列表并对它操作，使用 print(L)可以将整个列表内容输出。

接下来再通过一个例子练习一下列表的操作，假设有列表['a', 'b', 'c', 't', 'u', 'q', 'c', 'p', 'e', 'b']，其中有些项是重复的，我们要使用程序将重复项剔除。

程序 4.2　除去列表中的重复项：

```
list = ['a', 'b', 'c', 't', 'u', 'q', 'c', 'p', 'e', 'b']
re_list = []

for i in list:
    if not i in re_list:
        re_list.append(i)

print(re_list)
```

输出：

```
['a', 'b', 'c', 't', 'u', 'q', 'p', 'e']
```

分析：

这个程序的步骤比较明了，在遍历 list 的同时我们使用一个 if 语句过滤掉重复出现的项，最后将不包含重复项的新列表输出。

4.2　字　　典

字典就像它的名字一样，我们知道了关键字，就可以在字典里找到与关键字匹配的更多详细信息。字典在 Python 中是由一系列键-值对组成的，将键（keys）与值（values）关联到一起。注意，字典中的键必须是唯一的，而且，虽然键并不指定必须使用哪种类型，但是我们只能使用不可变对象作为键（如字符串）。对于字典中值的类型，并没有什么要求，我们可以根据具体约束选用可变或是不可变的对象。

如果说列表是有序对象的集合，那么字典就是无序对象集合。这种特性导致字典中元素需要通过键来存取，而不能像列表那样通过偏移存取。

接下来，通过化学课上学的元素周期表（见图 4.2）来看看字典的具体操作。

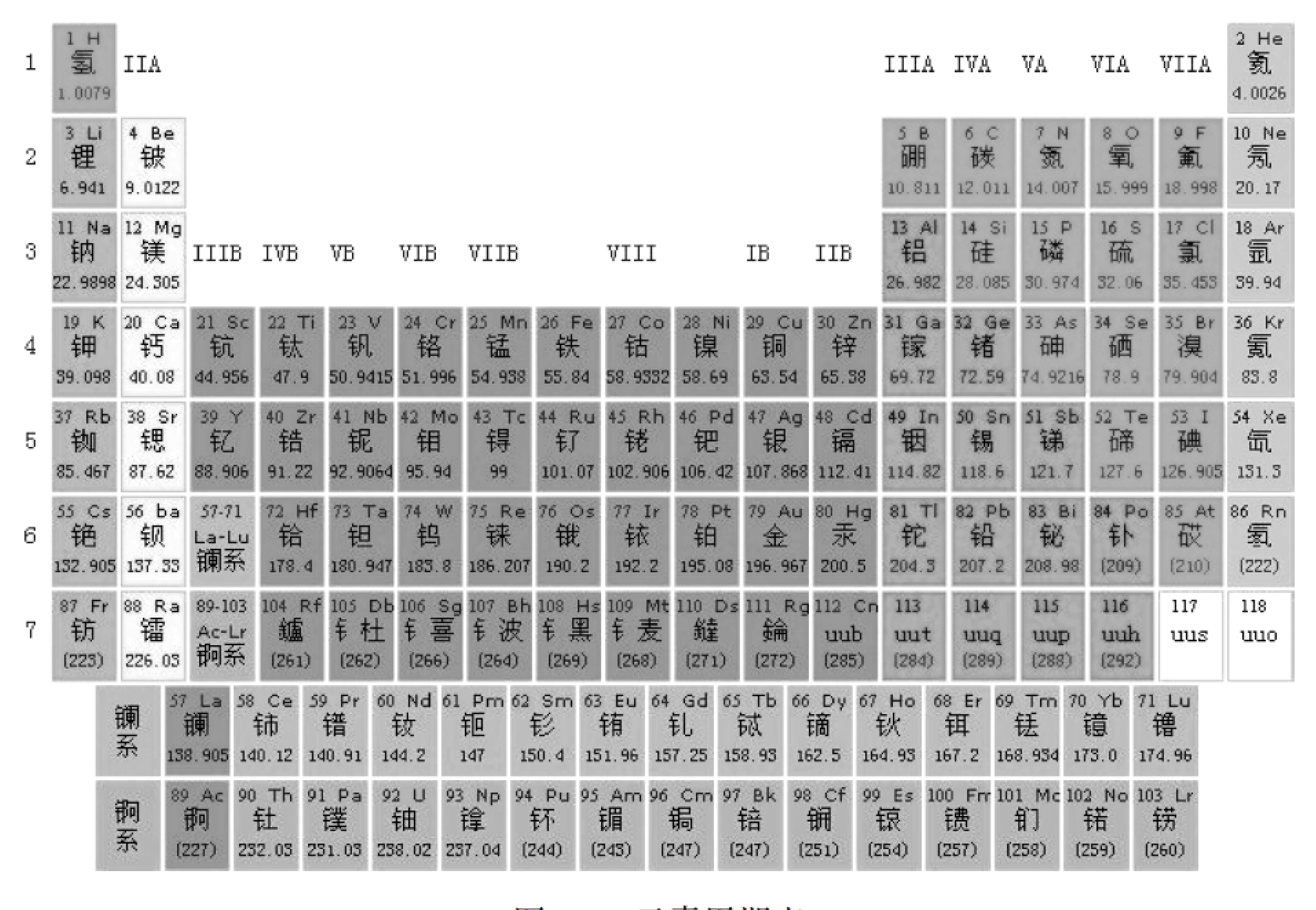


图 4.2　元素周期表

化学元素周期表对大家来说是很熟悉的，我们在初中就接触过。

在这里我们使用程序创建一个字典，字典中只存有元素周期表中的前 5 个元素。将元素的化学符号作为字典的键，元素名称作为值。具体程序如程序 4.3 所示。

程序 4.3　元素周期表：

```
1:  element = {'H': 'Hydrogen', 'He': 'Helium', 'Li': 'Lithium', 'Be': 'Beryllium', 'C': 'Carbon'}
2:
3:  del element['C']
4:  element['B'] = 'Boron'
5:
6:  print(element['H'])
7:
8:  for item in element.items():
9:      print(item)
```

输出：

```
Hydrogen
 ('H', 'Hydrogen')
 ('He', 'Helium')
 ('Li', 'Lithium')
 ('Be', 'Beryllium')
 ('B', 'Boron')
```

分析：

程序的第 1 行，我们使用：d = {key_1 : value1, key_2 : value2}这种形式来制定键值和值。注意，键值与值之间用“:”分开，每个元素（一组键值和值）使用“,”区分，最后将所有内容套在“{}”中构成一个字典。

通过图 4.2 的元素周期表，可以看到字典 element 中元素的顺序不对，我们在第 3 行删除键值 C 和与其对应的值。删除之后，在第 4 行添加键值 B 和值 Boron，添加完后，元素周期表的前 5 项已经在 element 中构建完成。

完成字典的构建之后，来看看如何输出字典的项。我们在程序的第 6 行，指定输出键值为 H 的值，即输出结果的第 1 行。最后，使用一个 for…in 语句迭代输出字典中的每组值。对于这个迭代语句我们要注意的是 element.items()语句中的 items()一定不要忘了，如果不加这项，只会迭代输出字典的键值。

表 4.2 中列出了一些具有代表性的字典操作。

表 4.2　常用的字典操作

序号	操作	解释
1	D = {}	将 D 声明为一个空字典
2	D = {'element': {'B': 'Boron'}}	将 D 声明为一个字典，并将字典{'B': 'Boron'}嵌套到 D 的值中
3	D = dict.fromkeys(['a', 'b'], 10)	通过指定键值创建字典 D，参数['a', 'b']是键值列表，参数 10 为默认值。其中键值列表也可以使用已声明的列表变量或是元组变量（后续介绍），例如，D = dict.fromkeys(list, 10)，创建完成后的 D 为{'a': 10, 'b': 10}
4	D = dict(zip(keys, vals))	可以先简单地将 zip 函数理解为将键值和值组合并返回一组有序对。keys 和 vals 可以是列表或是元组类型的数据。例如： keys = ['a', 'b'] vals = [11, 33,] D = dict(zip(keys, vals)) 此时 D 为{'a': 11, 'b': 33}
5	D = dict(H='Hydrogen', He='Helium')	字典的另一种定义方式，同使用 D = {'H':'Hydrogen', 'He':'Helium'}定义是一样的
6	D['Li']	返回字典 D 中键值'Li'对应的值
7	key in D	判断键是否在字典 D 中，键在字典中返回 True，否则返回 False
8	D.keys()	以列表的形式返回 D 中的所有键值
9	D.values()	以列表的形式返回 D 中的所有值
10	D.items()	以列表的形式返回 D 中可遍历的元素（键值和其对应的值）
11	D_1 = D.copy()	复制字典 D 并赋值给 D_1。注意：复制操作与将 D 赋值给某字典操作不一样，赋值操作是一种引用，而复制操作完成后，改变字典 D 的值不会对 D_1 造成影响
12	D.get(key, default)	返回字典 D 中与键值（key）对应项的值，如果指定的键值在 D 中不存在，返回默认值（default）
13	D.update(E)	将字典 E 中的元素（键值与其对应的值）添加到字典 D 中
14	D.pop(key)	将字典中与键值（key）匹配的元素删除并返回与该键值对应的值
15	len(D)	返回字典中元素的个数
16	D[key] = 42	通过指定字典 D 中的键值（key）更新该键值对应的值
17	del D[42]	通过指定字典 D 中的键值（key）删除该键值所在元素
18	list(D.keys())	以列表的形式返回字典 D 中的所有键值
19	D = {k : k*2 for k in range(4)}	通过 for 循环便捷地定义一个字典，生成的字典 D 为{0: 0, 1: 2, 2: 4, 3: 6}

对于表 4.2，每个操作都对应着一个解释，若是对表中的某些操作还不理解，可以在 PyCharm 中定义一个字典并对它操作，使用 print(D)即可输出各项内容。

在这里我们对表 4.2 再多说几句。表中使用 D、D_1、E 代表字典，key 表示字典中的键值，keys 和 vals 代表已经声明完成的键、值列表（或是元组）。

注意：

字典也是一种很灵活的工具，下面是使用字典的注意事项。

（1）字典是一种映射机制，一定要注意，它不是序列，也没有顺序的概念，所以不要对字典使用类似字符串和序列的那种切片操作。

（2）键不一定总是字符串。在例子中我们使用了大量的字符串作为键值，其实任何不可变对象都可以作为字典的键。例如，使用整数作为键值。

4.3　元　　组

元组是由简单的对象构成的，它和列表非常相似，但一定要注意的是元组是不可变的，它不支持任何方法的调用。也就是说我们不能编辑或改变元组。

关于元组的操作有很多和之前介绍的列表操作相类似，这里用程序结合程序的输出来看看具体如何操作。详细见程序 4.4 及其分析。

程序 4.4：

```
t1 = (0,)
t2 = (0, 'tuple', 17.1)
t3 = 0, 'tuple', 17.1
t4 = ('abc', ('list','dictionary'))
t5 = tuple('spam')

print(t2)
print(t3)
print(t4[1][1])
print(len(t3))
print(t1 + t2)
print(t1 * 3)
print('tuple' in t2)
print(t1.count(0))
```

```
15:    print(t3.index(17.1))
16:    print(list(t2))
17:
18:    for x in t3:
19:        print(x)
```

输出：

```
 (0, 'tuple', 17.1)
 (0, 'tuple', 17.1)
dictionary
3
 (0, 0, 'tuple', 17.1)
 (0, 0, 0)
True
1
2
[0, 'tuple', 17.1]
0
tuple
17.1
```

分析：

我们先来看看程序的前 5 行，每行都定义了一个元组。在第 1 行中即使只有一个元素，也要加上','，否则不能被视为元组。再看看第 2、3 行，通过输出结果的第 1、2 行可知它们的作用是一样的，即使第 3 行没有用括号将元素括起来。由于 t4 使用了嵌套，可以使用第 9 行的取值方式操作嵌套的内容，注意，我们不能更改元组中的内容（若是更改元组的内容会引发 TypeError: 'tuple' object does not support item assignment）。

程序的第 5 行使用了 tuple 函数将一个字符串类型转成元组，若是将函数的参数改为列表，它也可以将这个列表转为元组。

继续看程序的第 10～14 行，这些操作都是在前几个类型中很常见的操作了，这里不再多说。第 15 行使用元组的 index()方法，它可以返回参数在元组中的索引值。注意，若参数不在元组中将报出一个 ValueError。

通过输出结果的第 10 行可知，程序的第 16 行中的 list(t2)可以将元组转换成列表并返回。因为元组也是可迭代对象，我们在程序的最后使用 for…in 循环输出元组的项。

为什么需要元组?

学习列表和字典后，再学习元组可能觉得它没有使用的必要，为什么有了列表还需要元组？其实这正是因为元组特有的特性：不可变性。这种特点提供了一种完整性，它可以保证元组所在处不会被程序修改，这正是列表不具备的，而且这使得元组还可以作为字典的键值。总之，元组可以处理那些固定关系的问题。

接下来再用一个列子巩固一下之前讲过的内容。假设我们已经得到一篇小短文中的某 7 个单词的出现次数，需要根据这个次数找到出现次数最多的单词，接下来看看在 Python 中是如何实现的。具体程序如程序 4.5 所示。

程序 4.5　找到出现最多次的项：

```
max_value = 0
words = {'I': 4, 'He': 5, 'let': 9, 'be': 2, 'what': 3, 'how':2, 'the':7}
for item in words.items():
    if item[1] > max_value:
        max_value = item[1]
        word = item[0]

print("The most appeared word is : {}".format(word))
```

输出：

```
The most appeared word is : let
```

分析：

在程序中我们使用了一个 for 循环遍历字典，使用 item 获取一组键-值，但是，要注意的是 item 是一个元组类型的变量，就拿第一次循环来说，item 的值为('I', 4)。这是为了防止我们在使用字典的过程中修改了字典内容并导致程序产生错误。因此，可以使用元组加下标的方式使用 item 中的值用于比较，在这个过程中一定要注意的是不能改变元组的内容。在循环的过程中，我们使用 if 语句来选出最大值，并使用 word 保存最大值对应的键。最后将变量 word 输出。程序流程图如图 4.3 所示。

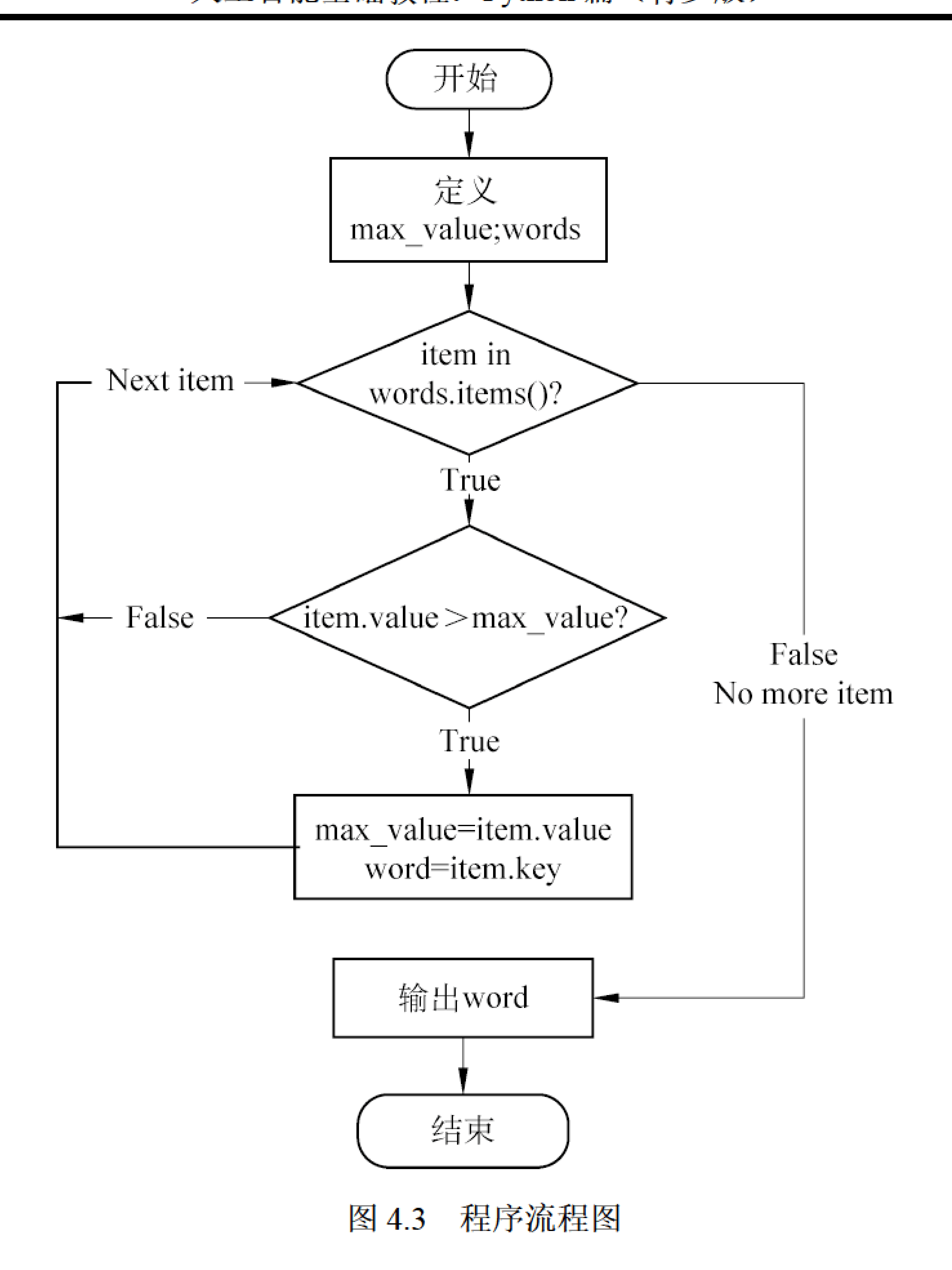


图 4.3　程序流程图

4.4　排序与查找

学习完列表、字典和元组后，我们趁热打铁，学习一下排序与查找的算法。

排序和查找是在实际工程中很常见的操作，它们有很多经典的方法。排序的目的是让数据能够以更有意义的形式表现出来，而查找的意义是在一个数据集中找到元素的位置。这两个内容可能有点难度，在这里我们不进行深入讲解，只对冒泡排序和二分查找这两个比较简单的算法来细谈。在完成这两个算法的学习后，将顺势引出算法复杂度的概念。

4.4.1　冒泡排序

冒泡排序是一种较为简单的排序算法，它会重复地访问要排序的数列，每次都比较两个元素，若是发现顺序有误就将它们交换过来。由于这个排序算法会很形象地将元素慢慢浮起（不断地两两交换），因此称为冒泡排序。

下面用一个例子来详细说明一下，我们构建一个乱序的化学元素列表（用元素在周期表中的序号作为排序的依据）['Cl', 'B', 'K', 'O', 'P']，为了便于编程，使用元素周期表的序号将这个列表抽象为[17, 5, 19, 8, 15]。准备好列表后，使用冒泡排序将这个列表进行升序排列。在给出具体的程序之前我们先用图 4.4 展示一下排序过程。

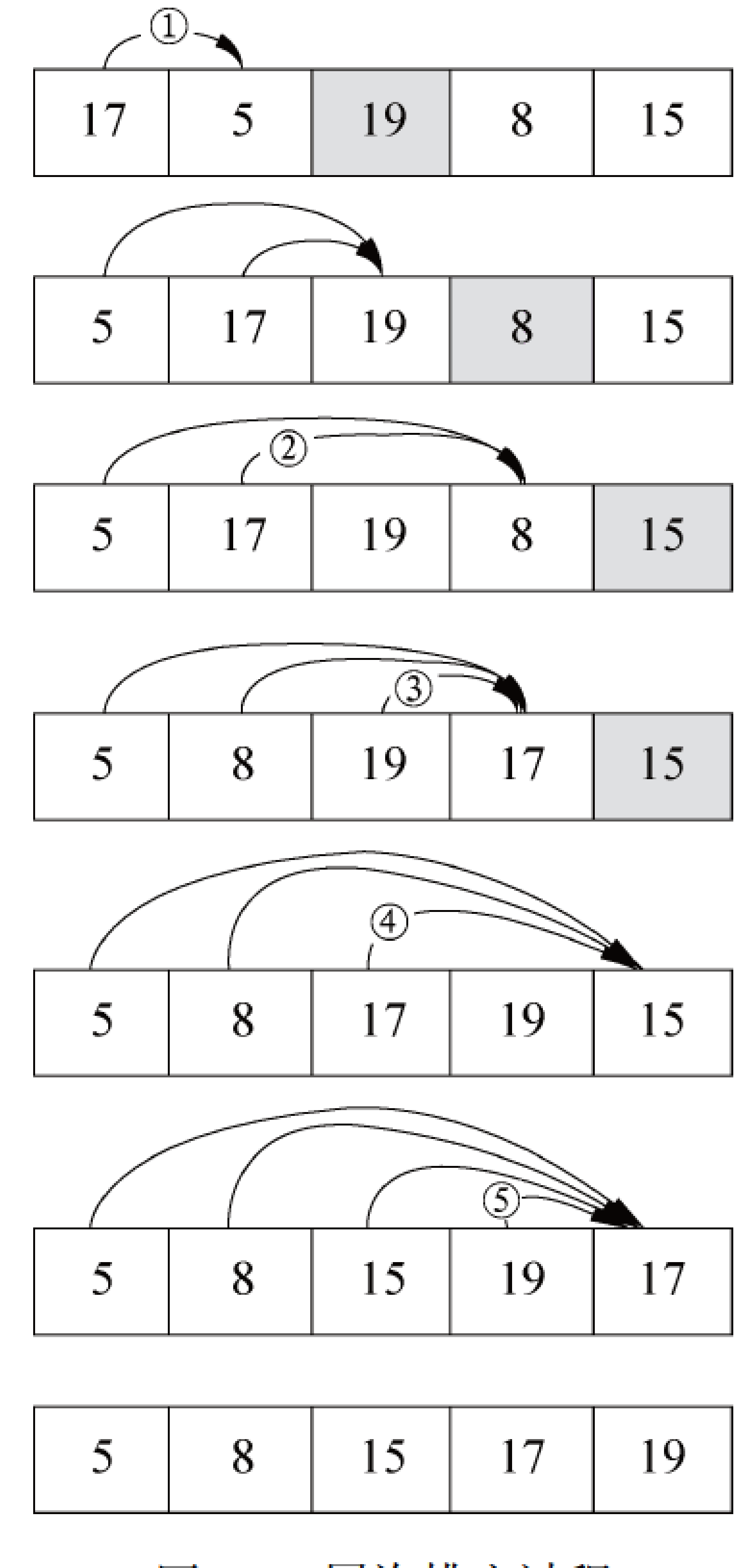


图 4.4　冒泡排序过程

图 4.4 中只给出了大致过程，其中阴影中选定项的两两互换过程我们并没有给出。在

图中的每行阴影选定的项，都会从头开始和同行中的第一个与阴影元素相邻的非阴影元素比较，顺序不对就交换。直到所有阴影元素都比较完，排序才完成。

可能这样说有点抽象，接下来给出具体的程序，让我们结合程序和图 4.4 再次分析。

程序 4.6　冒泡排序：

```
element = [17, 5, 19, 8, 15]

for i in range(len(element)):
    for j in range(i+1):
        if element[i] < element[j]:
            element[i], element[j] = element[j], element[i]
print(element)
```

输出：

```
[5, 8, 15, 17, 19]
```

分析：

在程序中我们使用了两个 for 循环嵌套，外层循环用于控制图 4.4 中右侧箭头的指向。内层循环用于从头开始不断选取箭头左边元素与外层所选的元素比较，若是顺序不对就将元素值调换。这样当外层循环结束，整个序列的排序工作完成。

4.4.2　二分查找

二分查找法作用于一个有序数据集合上。首先要查找的是有序集的中间的元素，如果中间元素比要查找的元素大，接着转向较小的半集中进行查找，反之，若中间元素比要查元素小，就转向较大的那个半集中进行查找。转进范围更小的数据集后重复这个查找过程，直到找到要查的元素或数据集不能再分割。

经过上面对二分查找的描述，我们对这个算法已经有所了解，二分查找法实质上就是不断地将有序数据集进行对半分割，并检查分区中的中间元素。上一小节中介绍了冒泡排序，正好可以使用冒泡排序的输出，也就是那个排好序的化学元素列表[5, 8, 15, 17, 19]。

假设想要知道氧元素在列表中的索引号，我们可以使用二分查找法来查找。下面，先来看看二分查找法是如何作用在这个有序的化学元素列表上的。详细过程如图 4.5 所示。

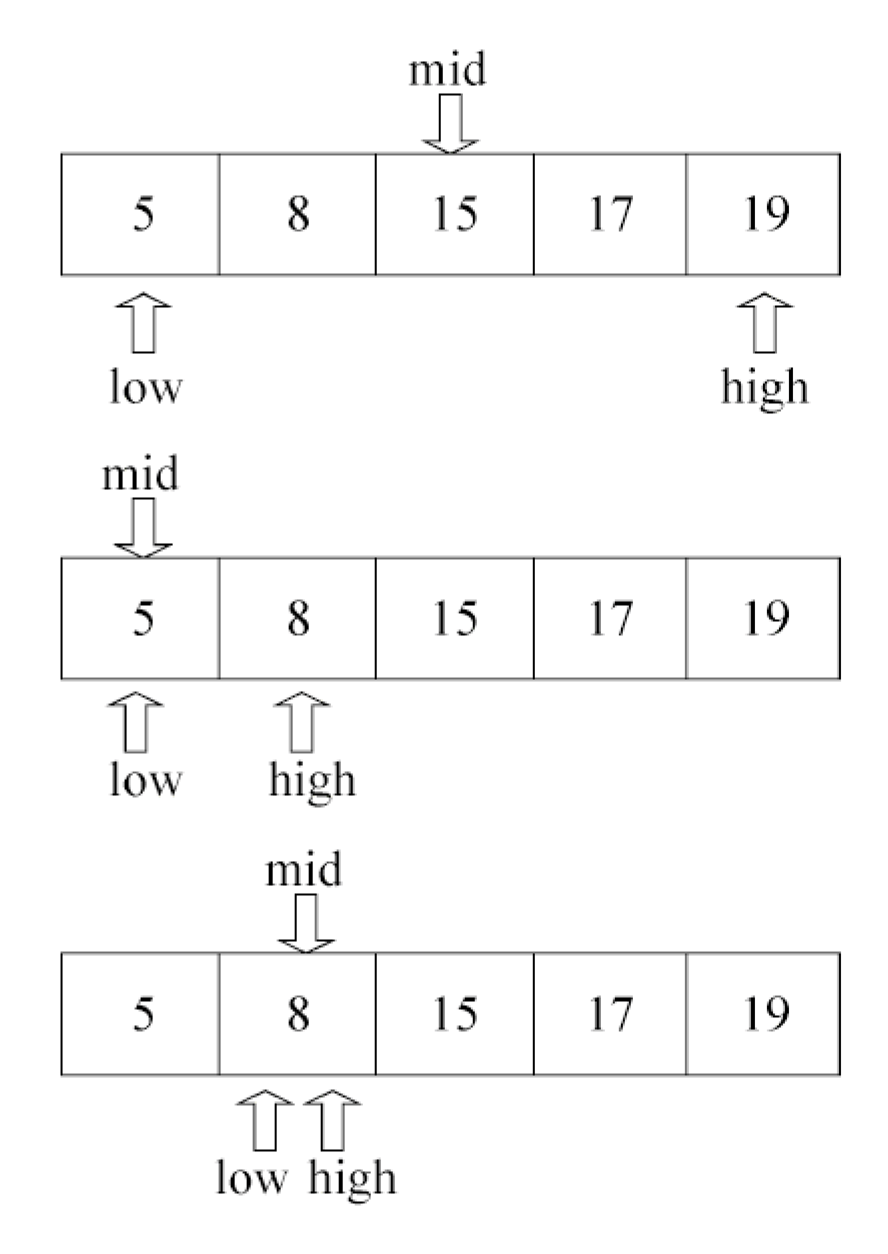


图 4.5　使用二分查找搜索

图 4.5 中的 low 和 high 是用来控制查找元素的两个边界值，它们圈起来的数据就是用于当前查找的数据集，mid 表示的是 low 和 high 之间的中间值。接下来看看程序具体是如何操作的，如程序 4.7 所示。

程序 4.7　二分查找：

```
element = [5, 8, 15, 17, 19]
val = 8
low = 0
high = len(element) - 1
trace = False
while low <= high:
    mid = (low + high) // 2
    if element[mid] == val:
        trace = True
        break
    elif element[mid] > val:
        high = mid - 1
    else:
        low = mid + 1

if trace:
```

```
    print("Find,the index of {0} is {1}".format(val, mid))
else:
    print("No element {0}".format(val))
```

输出：

```
Find,the index of 8 is 1
```

分析：

从程序中可以看出，我们将 low 和 high 分别设置为列表索引 0 和 len-1，在每次的 while 循环中将 mid 设置为 low 和 high 的中间值，若 mid 所指定的元素（element[mid]）比目标值小，将 low 移动到 mid 后的元素位置上，以此来更新搜索区间。同理，若 mid 所指定的元素比目标元素值大，将 high 更新到 mid 的前一个元素上。

随着循环不断推进，low 从左向右移，high 从右向左移，当 mid 处找到目标时，将 trace 标记为 True 并跳出循环。如果目标一直没找到，那 low 和 high 的指向将会重合并退出循环。这个过程也正是图 4.5 所展示的。

4.5　小酌算法分析

关于算法分析其实是一个偏后期的话题，相对于之前介绍的内容这一部分较为抽象，在这里我们只对其进行一些简单的理论介绍。本书中不管是示例的程序还是练习的程序，规模都非常小，每次运行的时候都是很快就出现结果，有些时候甚至是“秒出”，这是因为现在计算机技术的快速发展，即使是个人计算机也有着相对可观的计算能力，但是，这并不能改变运行程序时会消耗资源的事实。

在实际编程中，不管是设计还是应用一种算法，我们都要了解这种算法的性能如何。通常在衡量一个算法的性能时关心的是对 CPU 和内存的使用效率，但是由于 CPU 是计算机中最为珍贵的资源，我们还是以 CPU 的使用率为主，它体现在算法上便是算法的运算速度，用一个专业的词便是算法的时间复杂度。相应地，如果是估计程序需要的存储空间，要考虑的便是空间复杂度。

4.5.1　时间复杂度

由于不能对每个算法都上机测试，我们使用算法中语句的执行次数对算法进行估算，

哪个算法中语句执行次数多，它花费时间就多。同时，我们将算法中的语句执行次数称为时间频度，记为 $T(n)$，其中 n 表示语句的规模。

有了 $T(n)$之后，又出现一个问题，我们不知道随着处理数据量的增长它的变化会呈现出什么规律，因此引入辅助函数 $f(n)$的概念，它和 $T(n)$是同一量级的，同时，使用符号 $O(f(n))$来替代 $T(n)$，$O(f(n))$这种形式也被称为 O 表示法。这样我们就可以使用函数分析的手段较为准确地分析时间频度 $T(n)$，并以此来计算时间复杂度。

4.5.2　O 表示法的简单规则

由于有些运行处理对于当前算法来说影响太小了，我们对其选择忽略，具体的规则如下：

（1）常数项使用 $O(1)$表示。

（2）高阶因子和低阶因子并存，只保留高阶因子。

（3）如果最高阶项存在并且不是常数项，则去除它的系数。

对于上述的这些规则下面使用一些代码段进行详细分析。

4.5.3　算法分析示例

常阶数例子：

```
x = 1.0
y = 0.5
result = x / y
print(result)
```

上述一共 4 条语句，但是根据规则（1），它的时间复杂度为 $O(1)$，我们称之为常阶数项。具体来说，因为每条语句规模都是常数级的，这决定了它们整体是常数级的，就算有 10 条，100 条这样的语句也是常数级的，时间复杂度不会变。

线性阶例子：

```
for i in range(n):
    print('hello world')
```

因为循环体中的时间复杂度为 $O(1)$的代码执行了 n 次，这段代码的规模是由 n 的个数

决定的，因此，它的时间复杂度为 $O(n)$。

平方阶例子：

```
for i in range(n):
    for j in range(n):
        print('hello world')
```

由于这段代码中内层循环的次数是由外层决定的，因此，它的时间复杂度不能简单地设为 n^2，但是我们可以对其推导，代码中的输出语句执行次数为：$n+(n-1)+(n-2)+(n-3)+\cdots+1$。用等差数列求和得到其值为 $n^2/2+n/2$。根据规则（2）和规则（3）可得时间复杂度为 $O(n^2)$。

4.5.4　常见复杂度及其比较

下面来看看常见的算法复杂度及其例子，具体内容如表 4.3 所示。

表 4.3　常用的算法复杂度

序号	复杂度	例子
1	$O(1)$	从一个数据集中获取第一个元素
2	$O(\lg n)$	将数据集分为两堆，然后将分好的部分再分半，以此类推
3	$O(n)$	遍历一个数据集
4	$O(n \lg n)$	在做序号 7 的工作的同时遍历分出来的每一半数据
5	$O(n^2)$	遍历一个数据集中的每个元素的同时遍历另一个同数量级的数据集
6	$O(2^n)$	为一个数据集生成其可能的所有子集
7	$O(n!)$	给数据集生成所有的排列组合

当我们得到一个已知的情况的算法复杂度 $O(f(n))$时，使用数学曲线即可对其进行分析，假设现在要对两种已知的算法复杂度 $O(n^2)$和 $O(n \lg n)$进行分析，其中自变量为 n 也就是算法的规模，因变量为 $O(f(n))$或是 $T(n)$，先来看它们的函数曲线图，如图 4.6 所示。

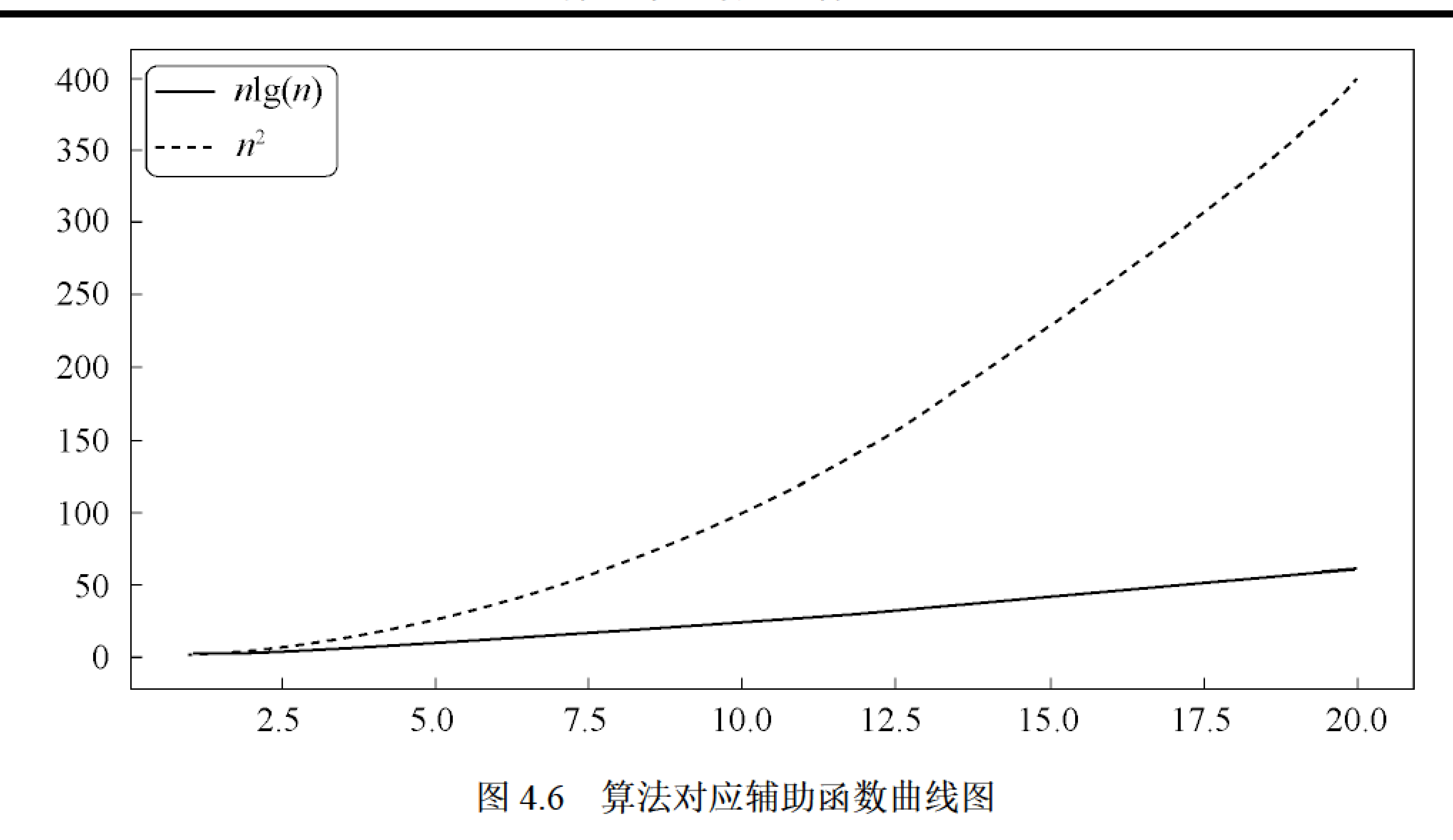


图 4.6　算法对应辅助函数曲线图

通过图 4.6 可以看到，问题规模（横轴数据）在 0～20 时，复杂度 $O(n^2)$会远大于 $O(n \lg n)$，因此，如果其他情况相同，我们选择复杂度为 $O(n \lg n)$的算法。

4.6　趣 味 练 习

本章的趣味练习要继续讲解第 2 章中使用过的 eval 函数，先来看程序 4.8。

程序 4.8：

```
import turtle
t = turtle.Pen()
turtle.bgcolor("black")

name_str = "['Leo', 'Lily', 'Tom', 'Alex', 'Max']"
name_list = eval(name_str)
colors = ["yellow","green", "white", "brown", "gray"]

for x in range(150):
    t.pencolor(colors[x % len(name_list)])
    t.penup()
    t.forward(x*4)
    t.pendown()
```

```
14:  t.write(name_list[x % len(name_list)], font=('Arial', int((x+4)/4)))
15:  t.left(360/len(name_list) + 3)
```

输出，如图 4.7 所示：

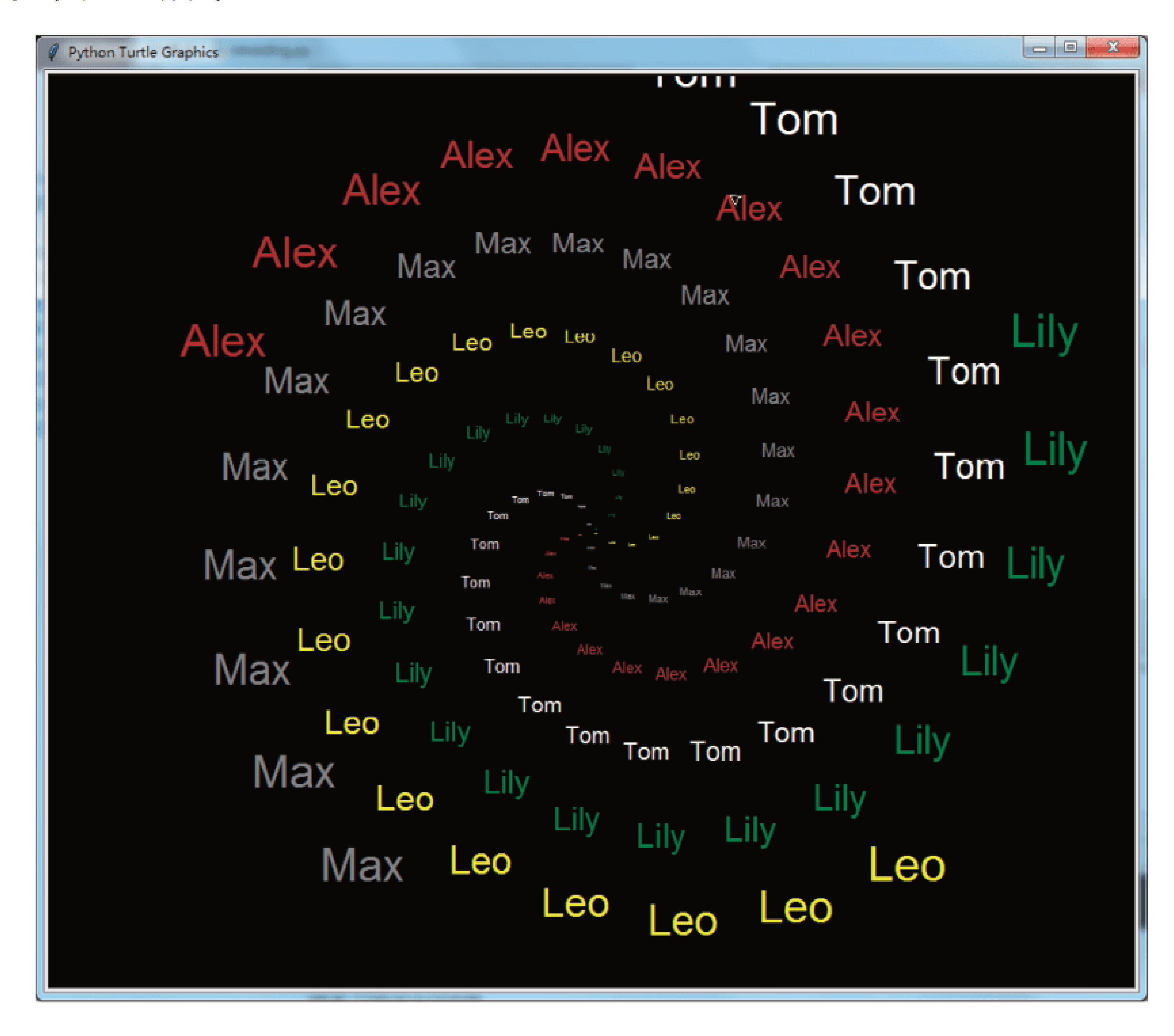


图 4.7　程序 4.8 输出结果

点睛：

程序的第 3 行，turtle.bgcolor()用来设置画布的颜色。第 6 行使用 eval 函数将整个字符串直接读成列表，此时的 name_list 的值为['Leo', 'Lily', 'Tom', 'Alex', 'Max']，它是一个列表，同时，我们还可以用 eval 函数将字符串中的字典或元组转换为用于程序处理的字典或元组。

程序的第 11 行和第 13 行的现实意义为画画的提笔和落笔动作，它在程序中的作用为将移动时的线条去除掉。程序中的 t.write 表示在画布中写下信息，参数 font=('Arial', int((x+4)/4))表示要写下信息的字体名和字号。

趣味一刻：

如果将程序的第 15 行内容改为 t.left(540/len(name_list) + 3)，会有什么效果呢？

4.7　总　　结

本章我们学习了 Python 中很关键的 3 个数据类型：列表、字典和元组。学习完这 3 个数据类型之后又简单地介绍了排序和查找算法。

列表、字典和元组都是 Python 中很常用的知识，需要我们尽快地掌握，这对于今后编写 Python 程序是至关重要的。关于排序和查找，这其实是一个比较大的知识块，我们只是对其中很简单的两个算法进行了介绍，希望大家可以熟练地使用。

4.8　练　　习

（1）观察下列程序，分析它的输出是什么，为什么？

```
element = {'H': 'Hydrogen', 'He': 'Helium', 'Li': 'Lithium', 'Be': 'Beryllium', 'C': 'Carbon'}
list = ['a', 'b', 'c', 't', 'u', 'q', 'c', 'p', 'e', 'b']
tuple = (0, 'tuple', 17.1)
print(list[1])
print(tuple[2])
print(element[1])
```

（2）我们能使用冒泡排序算法将元组中的数变为升序吗？为什么？

（3）用字典模拟一个列表，模拟内容为元素周期表中的前 10 项。

披荆斩棘（选做）：

编写程序对列表 [15, 11, 55, 88, 99, 47, 19, 11, 65, 1, 20, 84, 33] 排序，并查找有序表中值为 47 的元素位置。对程序做出简单的分析。

第 5 章　文 件 操 作

文件是计算机中具特定标识的存储区，它由操作系统管理，用于计算机操作系统的使用过程中的各项操作的支持。本章将讲解如何使用 Python 操控计算机中的文件，通过本章的学习，需要掌握：

- 如何从文件中读取数据。
- 如何向文件中写入数据。
- 大数据文件处理思想。

5.1　文件及其操作

对于文件大家都不陌生，但是在接触到计算机的文件之前，我们通常将文件定义成内容的载体，如公文书信或是有关政策理论等方面的文章。计算机文件也一样，它是存储在计算机存储区的信息集合，这些信息有很多用途，有的是用来支撑程序的运行，有的是单纯用于存储等。文件使用文件扩展名指示文件类型，如常用的 JPEG 格式的图像文件使用.jpg 文件扩展名。

由于我们将大段的待处理信息存储在文件中，当处理这些信息时需要通过 Python 来调用这些信息进程序，然后在程序中对其进行处理。通过这种方式使得程序可以处理任何指定位置的文件。流程如图 5.1 所示。本章学习对文件的操作主要是写信息到文件和从文件中读取信息。

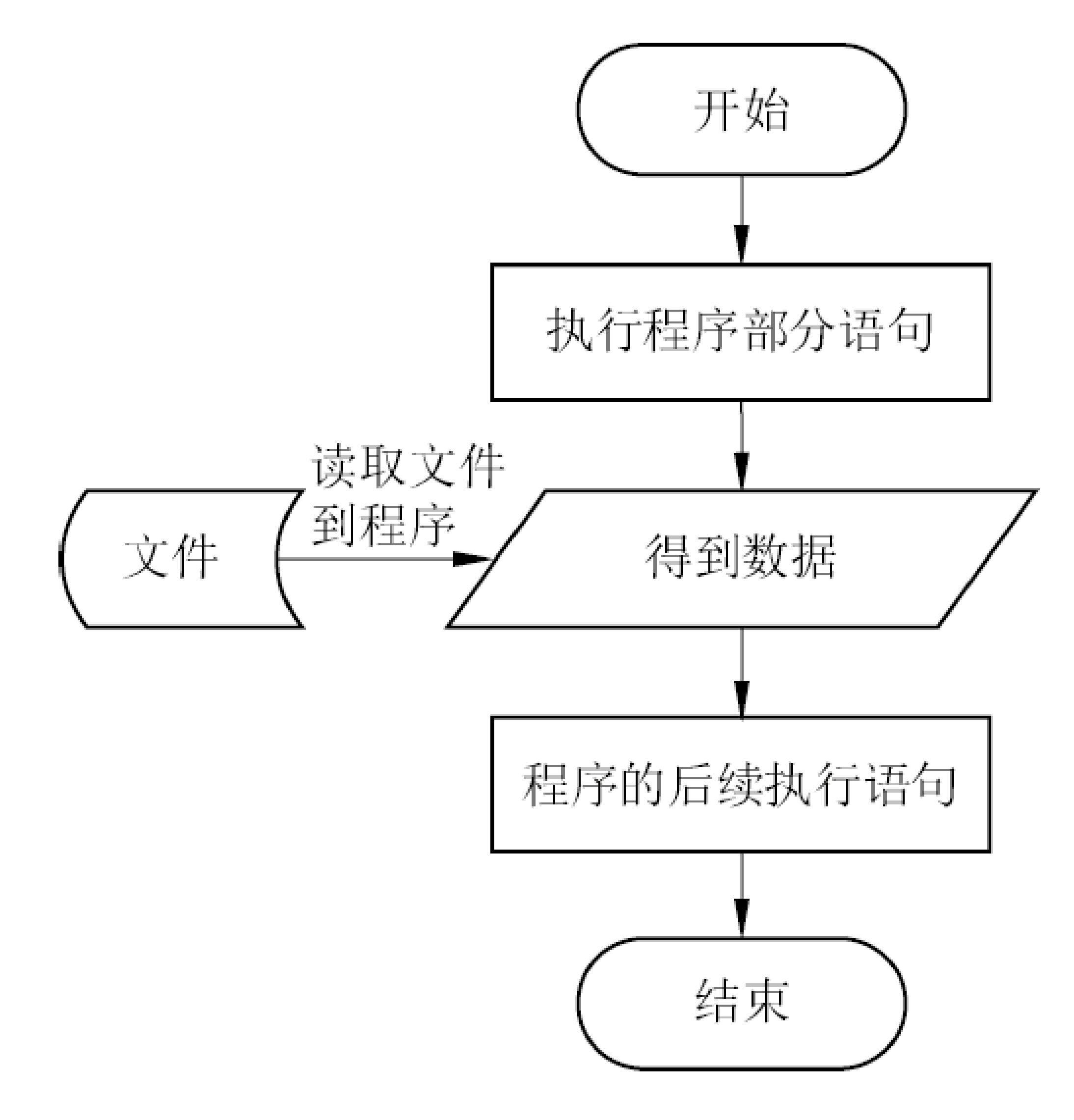


图 5.1　程序处理文件内信息步骤

5.2　从文件中读取数据

本节将介绍使用 Python 程序读取文件的具体方法。Python 程序想要使用计算机存储区的文件时，要遵从 Python 文件操作的规定，依顺序进行读取，若是可以随便处理会使文件内容变得很不安全。

简单地说，想要在 Python 程序中读取文件，先要使用 Python 内置的 open 函数通过提供文件路径的方式将文件和程序链接起来，之后便可以通过操作文件对象的方法处理文件。接下来通过程序 5.1 来看看读文件操作是如何进行的。

程序 5.1：

```
file = open('C:/Users/dell.dell-PC/Desktop/testfile.txt')
while True:
    line = file.readline()
    if len(line) == 0:
        break

    print(line, end="")
file.close()
```

输出：

```
this is the first line
and the second
the end
```

分析：

程序 5.1 十分清晰地展现了文件操作中打开文件，对文件操作，关闭文件的步骤。程序的第 1 行我们使用了之前提到的 open 函数建立了一个 file 对象，程序的第 3 行使用了 file 对象的 readline 方法按行读取文件，当再次调用 readline 时会自动跳到文件的下一行。程序的第 8 行使用 close 方法将文件关闭，其实这个操作是可选的，因为 Python 中一旦对象不再被引用，则这个对象的内存就会被自动回收。但是从另一方面来说，手动调用没有任何坏处，并且，随着程序越做越大，这是一个很好的习惯。

上面是一个很简单的程序，但是 Python 给我们提供的文件操作并非如此简单。表 5.1 列出了有关 Python 读取文件中的其他常见操作。

表 5.1　常见的读取文件操作

序号	操作	解释
1	file = open("path", "r")	path 指文件目录，语句同程序 5.1 的第 1 行，"r"表示读入
2	string = file.read()	将这个文件读入一个字符串
3	string = file.read(N)	从文件当前位置读取之后的 N 个字节到字符串
4	list = file.readlines()	将文件按行读到列表中，程序 5.1 中的文件使用这条语句 list 的值为['this is the first line \n', 'and the second\n', 'the end']
5	for line in open("path")	迭代一行行地读取

5.3　写数据到文件

上一节中学习了读文件内容到程序，接下来介绍一下写数据到文件。写数据的顺序其实和读文件差不多，具体为建立文件链接，写数据和关闭文件。下面通过程序 5.2 来看看写数据到文件是如何操作的。

程序 5.2：

```
1:    poem = '''Beautiful is better than ugly.
2:    Explicit is better than implicit.
```

```
3:    Simple is better than complex.
4:    Complex is better than complicated.
5:    Flat is better than nested.
6:    ...
7:    '''
8:
9:    file = open("C:/Users/dell.dell-PC/Desktop/testWriteFile.txt", "w")
10:   file.write(poem)
11:   file.close()
```

输出到文件的内容：

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
...
```

分析：

程序 5.2 的输出是程序的第 9 行中路径指定的 txt 文件内容，本程序其实在 Python 解释器中没有输出。程序的第 1～7 行使用三引号'''…'''给变量 poem 指定了带有换行的字符串，第 9 行使用带有目录的 open 函数指定具体的文件，同时给本次操作指定操作模式为 w。操作模式 w 表示打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。第 10 行使用 write 函数将 poem 写入文件，最后将文件关闭。此时写入文件已经完成，在对应目录下会出现该文件。

程序中出现的 write 函数是一个很常见的函数，表 5.2 列出了有关写入文件的另一些常见用法。

表 5.2　常见的写入文件操作

序号	操作	解释
1	file = open("path", "a")	如果该文件已存在，文件指针放在结尾用于追加。新内容会被写入到已有的内容之后。若文件不存在会先创建
2	file.writelines(list)	将列表中所有字符串写入文件中
3	file.seek(N)	将文件指针设置到当前向后加 N 处用于下一次操作

5.4　从 Web 页面读数据

本节中我们使用 Python 来做一个简单的读取页面信息的程序。在这个程序中使用了 BeautifulSoup 和 requests 两个模块。在使用这两个模块之前可能要先进行安装。这里先列出安装方法：使用 Win+R 组合键打开“命令提示符”窗口，输入 pip install requests 安装 requests 模块，输入 pip install beautifulsoup4 安装 BeautifulSoup 模块。完成后就可以使用这两个模块来快速构建一个读取 Web 页面信息的程序了，具体程序如程序 5.3 所示。

程序 5.3：

```
from urllib import request
from bs4 import BeautifulSoup

url="http://www.jianshu.com"
headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36'}
page = request.Request(url,headers=headers)
page_info = request.urlopen(page).read().decode('utf-8')

soup = BeautifulSoup(page_info, 'html.parser')

titles = soup.find_all('a', 'title')

file = open("C:/Users/dell.dell-PC/Desktop/webinfo.txt","w")
for title in titles:
    file.write(title.string + '\n')
    file.write("http://www.jianshu.com" + title.get('href') + '\n\n')
```

输出到文件内容（部分）：

```
[<a class="title" href="/p/3ea8262b0927" target="_blank">比贫穷更可怕的，是缺乏这 3 样东西</a>, <a class="title" href="/p/a64529b4ccf3" target="_blank">刘若英把《后来》拍成电影，你怎么看哭了？</a>, <a class="title" href="/p/ce340277fc6e" target="_blank">你为什么至今单身？看完后扎心了！</a>, <a class="title" href="/p/22c5b6081eac" target="_blank">《后来的我们》：无法跟喜欢的人在一起……]
```

分析：

在对程序进行分析之前先强调一下，输出只是部分内容，程序成功执行之后会在指定

目录出现相应的文件。同时，本节内容算是为后续内容进行铺垫，在这里我们只需简单了解处理步骤，本书后文会详细地讲解 Python 处理网络的相关知识。

程序的前两行将要用到的 requests 和 BeautifulSoup 两个模块中的内容引入。requests 模块用于从指定的网址上获取信息，BeautifulSoup 模块用于从获取的信息中提取结构化数据。程序的第 4～7 行，将想要获取信息的网址给变量 url，同时将头文件给变量 headers 用于模拟浏览器访问。构造之后将这两个变量用于 Request()方法并返回网站页面的请求对象 page，通过这个对象得到页面信息 page_info。

程序的第 9 行得到信息之后使用 BeautifulSoup 函数将其转化为 BeautifulSoup 格式同时指定 html.parser 作为解析器。信息转换完成之后存在变量 soup 中。

程序的第 11 行，提取 soup 中的所有 a 元素（表示 Web 页面中的链接部分）中 class='title' 的部分，即链接中的标题部分。完成之后将其输出，此时我们再看程序的输出，是一个列表，它的格式还是原始的 HTML（超文本标记语言，用于静态网页的制作）格式。

程序的第 13 行我们使用 open 函数创建写入文件的 file 对象用于将信息输入文件中，后面部分是迭代输入 titles 列表的各项内容，其中有一个细节要注意，对应输出来看，列表中各项的形式为string，这是 HTML 中表示链接的形式，<a>和称为开始和结束标签，<a>和标签中间夹着的是链接标题的内容，href 的内容是 a 元素内容的跳转地址。通过这段介绍再看程序的第 15、16 行的内容就会变得很明了，使用 title.string 获得 a 元素的标题内容，使用 title.get('href')得到 a 元素内容的跳转地址。由于网站在不断地更新，不同时间获取同网站的内容也会不相同，本次运行得到的部分信息如图 5.2 所示。

```
比贫穷更可怕的，是缺乏这3样东西
http://www.jianshu.com/p/3ea8262b0927

刘若英把《后来》拍成电影，你怎么看哭了？
http://www.jianshu.com/p/a64529b4ccf3

你为什么至今单身？看完后扎心了！
http://www.jianshu.com/p/ce340277fc6e

《后来的我们》：无法跟喜欢的人在一起，其实是人生的常态
http://www.jianshu.com/p/22c5b6081eac
```

图 5.2 获取到的网站信息（部分）

5.5　浅谈 Python 处理大数据文件

虽然相比于 C++等编程语言，使用 Python 处理大数据文件效率不高，但是由于 Python 开发速度快、代码量少、易于维护、成本低，并且有些细节问题使用 Python 处理极为方便。因此，在很多情况下会选用 Python 来处理大数据文件。这里介绍一下使用 Python 处理的两种方法：

（1）将文件切分为多个小段，同时处理多段，处理完成后将处理结果合并。

（2）使用 Python 自带的迭代器分行处理文件。

由于这个问题难度较大，我们在这里只列出处理思想，不给出具体的程序案例。其实上述两种方法涉及的是一种名为分治法的经典算法，算法的流程图解如图 5.3 所示。通过这种处理可以充分利用现有的计算资源，但是同时带来的是对于问题的分解管理。具体来说，分治法就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单地直接求解。最后，原问题的解就变成子问题解的合并。算法思想比较简单，但是真正处理时会有很多细节需要注意。

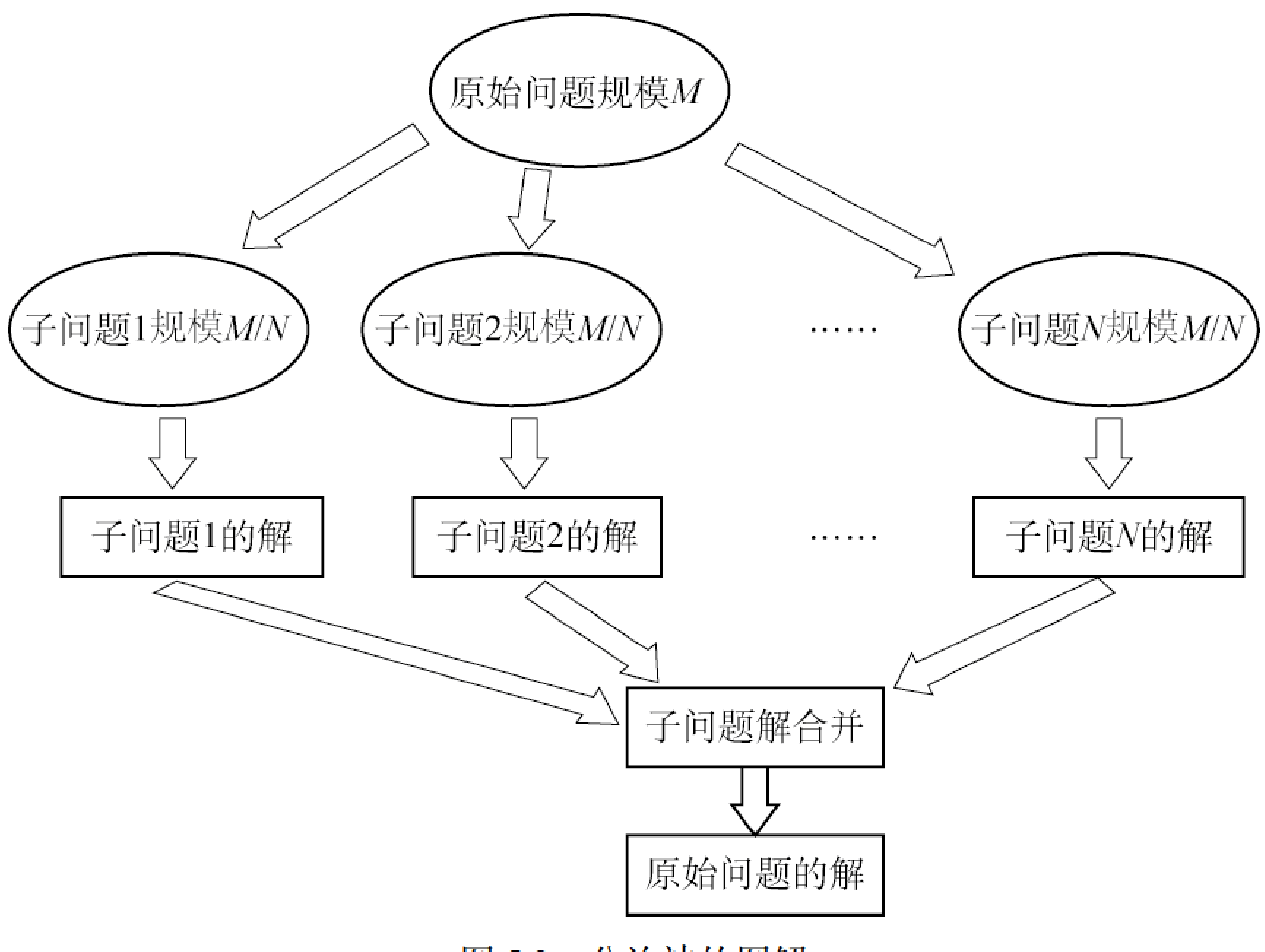


图 5.3　分治法的图解

5.6　案例：计算文件中关键字出现次数

本节将使用 Python 读取一个文件，统计某个特定字符串出现的次数并将其保存在某个文件中。本次我们使用 collections 模块，若是程序报出没有找到该模块的错误，请使用 pip 自行安装。

collections 模块是在 Python 内置数据类型之上，提供了多个有用的集合类模块。由于要统计特定字符串的出现次数，我们选用模块中的 Counter。关于程序具体的实现参考程序 5.4。

程序 5.4：

```
import collections

file = open('C:/Users/dell.dell-PC/Desktop/test_file.txt')
str = file.read().split(' ')
n = collections.Counter(str)
print(n['the'])

s = zip(n.values(), n.keys())

output = open('C:/Users/dell.dell-PC/Desktop/result.txt','w')
for item in sorted(s, reverse=True):
    output.write("{0} {1}\n".format(item[1], item[0]))
```

输出：

```
9
```

输出到文件内容（部分）：

```
to 11
and 10
the 9
thank 6
for 6
would 5
like 5
my 4
```

```
book 4
I 4
```

分析：

程序 5.4 充分体现了 Python 的开发速度快、代码量少的特点，由于 Counter 的使用，我们只需要提供数据。程序的第 4 行 str = file.read().split(' ')将其分解介绍，file.read()在前文中讲过，它是将文件中内容以一个字符串读出，此时语句相当于 string.split(' ')，而对于 split(' ')它是作用在一个字符串上的用于将字符串按指定规则（即它的参数，本次程序中指定为空格）分割，并将分割后内容以一个列表返回。

至此，str 中为['I', 'would', 'like', 'to', 'thank', 'everyone', 'at', 'No', 'Starch', 'Press', 'for'…'favorite', 'books.\n']。接下来这条语句可以说很精彩了，collections.Counter(str)会返回一个字典，我们使用变量 n 接收。其中键为 str 中的内容，值为每项出现的次数，具体内容为{'to': 11, 'and': 10, 'the': 9, 'I': 4, 'thank': 6, 'for': 6, 'would': 5, 'like': 5, 'book': 4, 'my': 4, 'everyone': 3, 'this': 3…'books.\n': 1}，关键字及其出现的次数已经统计完成，有没有感受到 Python 的强大！得到字典 n 之后，我们使用 n['the']取出文中'the'出现的次数。

完成这些处理之后我们将字典 n 中的键和键值单独抽出，使用 zip(n.values(), n.keys())将这些数据压缩成列表变量 s。

完成数据采集后开始为数据写入文件做准备。在打开文件后，我们使用 sorted(s, reverse=True)，以数据中的 n.values()值为标准对数据进行一个倒序的排列操作，最后使用“字符加空格再加字符出现的次数”这种格式将数据写入文件。

到这里程序已经结束了，你可能会有疑问，明明只要将数据写入文件就好了，为什么要先排序，再按照固定格式写入文件？这样处理会为后续操作带来极大的便利。我们先不讨论这个问题，但是在这里要说的是任何努力都不会白费的。

5.7 趣 味 练 习

本章的趣味编程我们来谈谈使用 Python 处理 JSON 格式的文件。

我们都知道，文件格式是计算机为了存储信息而使用的对信息的特殊编码方式，本节就来详细谈谈 JSON 格式。这一章的趣味编程可能不像之前的趣味编程那样有趣，但是，对于编程而言，JSON 格式是一个很有代表性的通信格式。JSON 格式文件内容如图 5.4 所示。

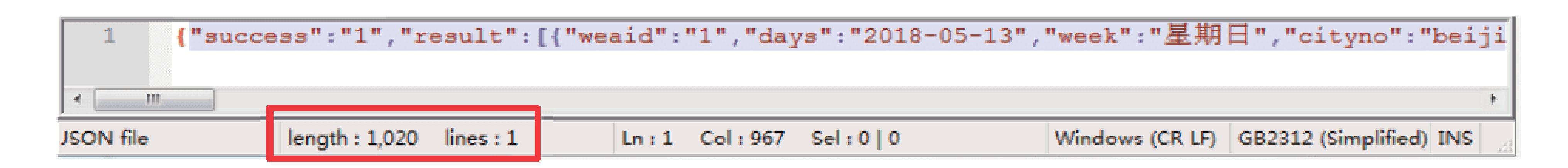


图 5.4　JSON 格式文件内容

图 5.4 中的 JSON 格式文件内容是一段没有加任何处理的 JSON 信息段，从方框中的内容可以看出，它只有一行。对于这种格式我们应该怎样处理呢？详见程序 5.5。

程序 5.5：

```
1:   #{"success":"1","result":[{"weaid":"1","days":"2018-05-13","week":" 星 期 日 ","cityno":"beijing","
citynm":" 北 京 ","cityid":"101010100","temperature":"30 ℃ /17 ℃ ","humidity":"0%/0%","weather":" 晴
","weather_icon":"http://api.k780.com/upload/weather/d/0.gif","weather_icon1":"http://api.k780.com/upload/
weather/n/0.gif","wind":" 南 风 ","winp":"3-4 级 转 <3 级 ","temp_high":"30","temp_low":"17","humi_
high":"0","humi_low":"0","weatid":"1","weatid1":"1","windid":"5","winpid":"402","weather_iconid":"0","
weather_iconid1":"0"},{"weaid":"1","days":"2018-05-14","week":" 星 期 一 ","cityno":"beijing","citynm":"
北 京 ","cityid":"101010100","temperature":"34 ℃ /22 ℃ ","humidity":"0%/0%","weather":" 晴 转 多 云
","weather_icon":"http://api.k780.com/upload/weather/d/0.gif","weather_icon1":"http://api.k780.com/upload/weat
her/n/1.gif","wind":" 南 风 ","winp":"<3 级 ","temp_high":"34","temp_low":"22","humi_high":"0","humi_low":"
0","weatid":"1","weatid1":"2","windid":"5","winpid":"395","weather_iconid":"0","weather_iconid1":"1"}]}
2:
3:   import json
4:   with open('C:/Users/dell.dell-PC/Desktop/test_json.json') as file:
5:       data = json.load(file)
6:
7:   print(data["success"])
8:   print(data["result"][0]["citynm"],
     data["result"][0]["days"], data["result"][0]["temperature"])
9:   print(data["result"][1]["citynm"],
     data["result"][1]["days"], data["result"][1]["temperature"])
```

输出：

```
1
北京 2018-05-13 30℃/17℃
北京 2018-05-14 34℃/22℃
```

点睛：

为了方便大家查看，在程序的第 1 行我们将程序中使用的 JSON 信息段以注释的形式列出，它也是图 5.4 的内容。我们将其存入 test_json.json 文件中用于程序的处理。

程序首先引入了用于JSON格式处理的json模块，接下来，使用 data = json.load(file)将 JSON 格式文件解析，此时，JSON 格式文件中内容的格式已经有了变化，详细见图 5.5 中的内容。


```
{
    "success": "1",
    "result": [{
            "weaid": "1",
            "days": "2018-05-13",
            "week": "星期日",
            "cityno": "beijing",
            "citynm": "北京",
            "cityid": "101010100",
            "temperature": "30℃/17℃",
            "humidity": "0%/0%",
            "weather": "晴",
            "weather_icon": "http://api.k780.com/upload/weather/d/0.gif",
            "weather_icon1": "http://api.k780.com/upload/weather/n/0.gif",
            "wind": "南风",
            "winp": "3-4 级转<3 级",
            "temp_high": "30",
            "temp_low": "17",
            "humi_high": "0",
            "humi_low": "0",
            "weatid": "1",
            "weatid1": "1",
            "windid": "5",
            "winpid": "402",
            "weather_iconid": "0",
            "weather_iconid1": "0"
        },
        {
            "weaid": "1",
            "days": "2018-05-14",
            "week": "星期一",
            "cityno": "beijing",
            "citynm": "北京",
            "cityid": "101010100",
            "temperature": "34℃/22℃",
            "humidity": "0%/0%",
            "weather": "晴转多云",
            "weather_icon": "http://api.k780.com/upload/weather/d/0.gif",
            "weather_icon1": "http://api.k780.com/upload/weather/n/1.gif",
            "wind": "南风",
            "winp": "<3 级",
            "temp_high": "34",
            "temp_low": "22",
            "humi_high": "0",
            "humi_low": "0",
            "weatid": "1",
            "weatid1": "2",
            "windid": "5",
            "winpid": "395",
            "weather_iconid": "0",
            "weather_iconid1": "1"
        } ]
}
```


图 5.5　解析后的 JSON 格式文件数据

怎么样，是不是有了一种豁然开朗的感觉！原本乱糟糟的内容其实是一段天气预报信息。此时再来看程序的第 7～9 行，这种使用方法有点像 Python 中列表、字典的访问形式，例如，图 5.5 中有{}包住的数据，也有[]包住的数据，其中使用[]包住的数据需要使用列表的访问形式进行访问，如[1]这种形式。而{}包住的数据需要使用像字典的键访问键值的形式来提取，如["days"]这种形式。将这两种形式进行组合便可以达到提取所有信息的目的。

5.8　总　　结

通过本章的学习，我们看到 Python 中具体的文件操作，还学习了较为抽象的分治思想。关于文件操作在实际的编程中可能用到的不是特别多，但它是一个很重要的工具，例如，我们可以将程序的运行状态写到文件中并查看文件内容的方式，监视程序的运行情况；将程序处理的数据写入文件中，并用更为专业的结果分析工具打开文件查看程序执行情况等。

5.9　练　　习

（1）open 函数默认处理模式是什么？

（2）将列表[5,6,7,8,2,5,6,3,7,2]的内容按一行一个元素写入文件中，再将文件中内容去除并排序，完成后再次写入文件。

（3）自己找到一个有 200 行内容的文件，提取第 77、90、144 行的内容，要求编写两个程序来模拟分治法的执行。

第 6 章　绘制需要的图表

本章将使用 Python 中的 pandas 模块进行数据分析并使用 matplotlib 模块根据数据绘图。对于这两个模块在本章中会给出介绍并将其应用于案例中。通过本章的学习，需要掌握：

- matplotlib 模块和 pandas 模块的基础知识。
- matplotlib 模块和 pandas 模块的联合使用。
- 常用图形的画法。
- 将画图部分应用到程序中。

6.1　matplotlib 基础

本章我们将在 PyCharm 中使用 matplotlib 模块来编写程序，在使用前，同之前一样在“命令提示符”窗口中使用命令 pip install matplotlib 安装模块。也可以在编写程序时安装，假设我们没有引入模块，并在 PyCharm 中使用语句 import matplotlib.pyplot as plt，PyCharm 会提示错误，此时将光标移动到 import 语句上的任何位置会显示一个小灯泡的图标，这时再单击图标，选择 install package matplotlib 选项，在这之后 PyCharm 会帮我们完成安装。此过程如图 6.1 所示。

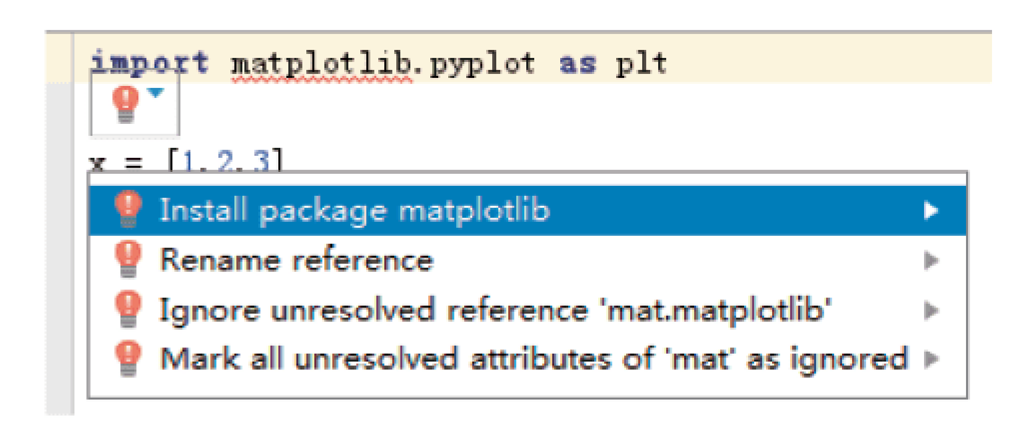


图 6.1　在 PyCharm 中引入模块

关于 import matplotlib.pyplot as plt 语句，可以理解成将 matplotlib 模块的 pyplot 以新名字 plt 导入到要编写的程序中。完成后先来看下面这段程序。

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4, 5], [1, 4, 9, 16, 25])
plt.show()
```

程序中使用 plot()方法绘制坐标，坐标（*x*, *y*）中 *x* 和 *y* 的值分开存在列表中，并按先 *x* 后 *y* 的顺序将参数传入 plot()方法中。程序运行结果如图 6.2 所示。

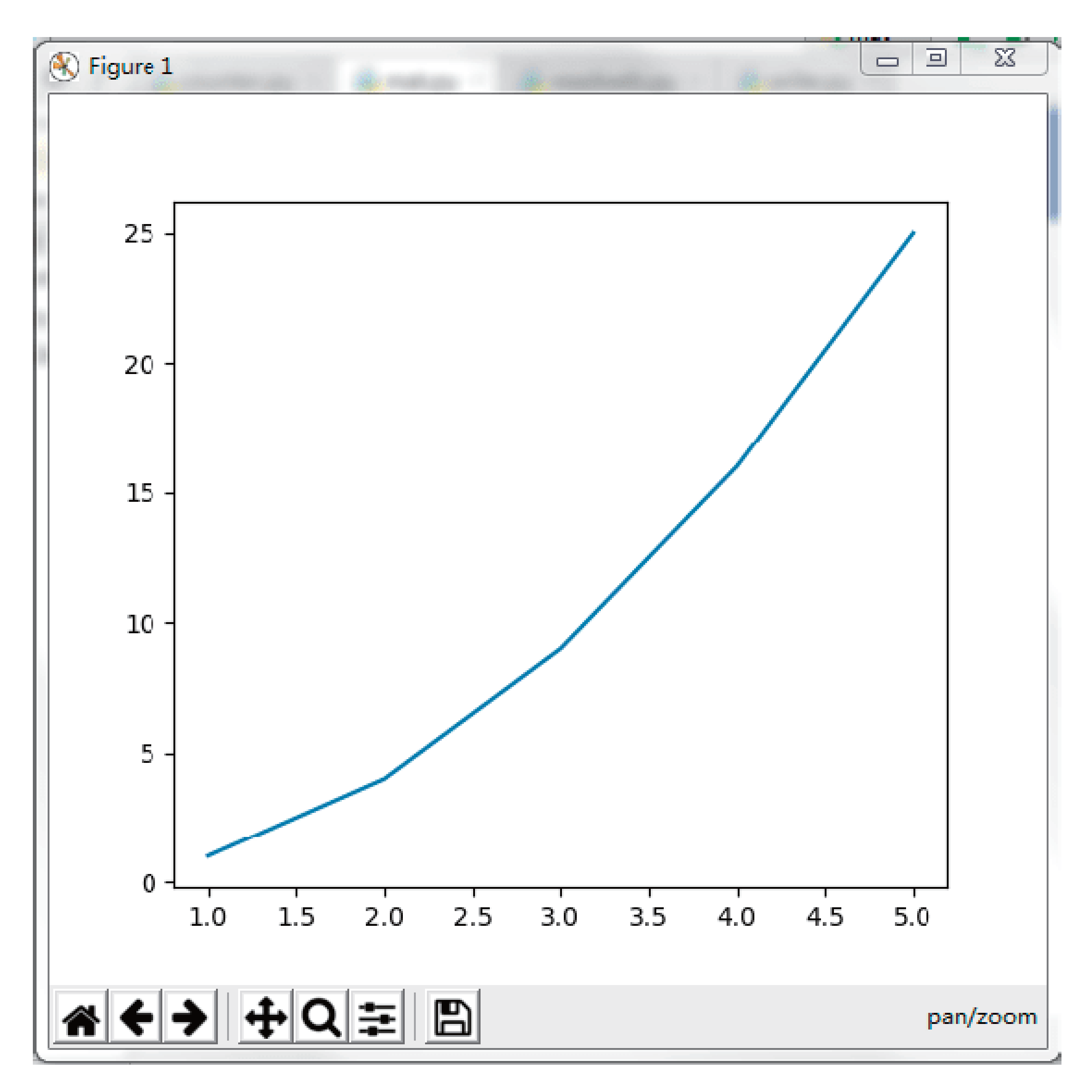


图 6.2　程序运行结果

在 PyCharm 中运行可能并不会单独弹出窗口，这是 PyCharm 的设置问题，可以通过取消选中 File→Settings→Tools→Python Scientific 中的 Show plots in toolwindow 并单击右下角的 Apply 按钮完成设置。这时再运行程序就会出现如图 6.2 所示的窗口。

通过 matplotlib 显示的窗口不仅可以查看图形，还允许我们与其进行交互。它在左下角处提供了 7 个按钮，如图 6.3 所示。

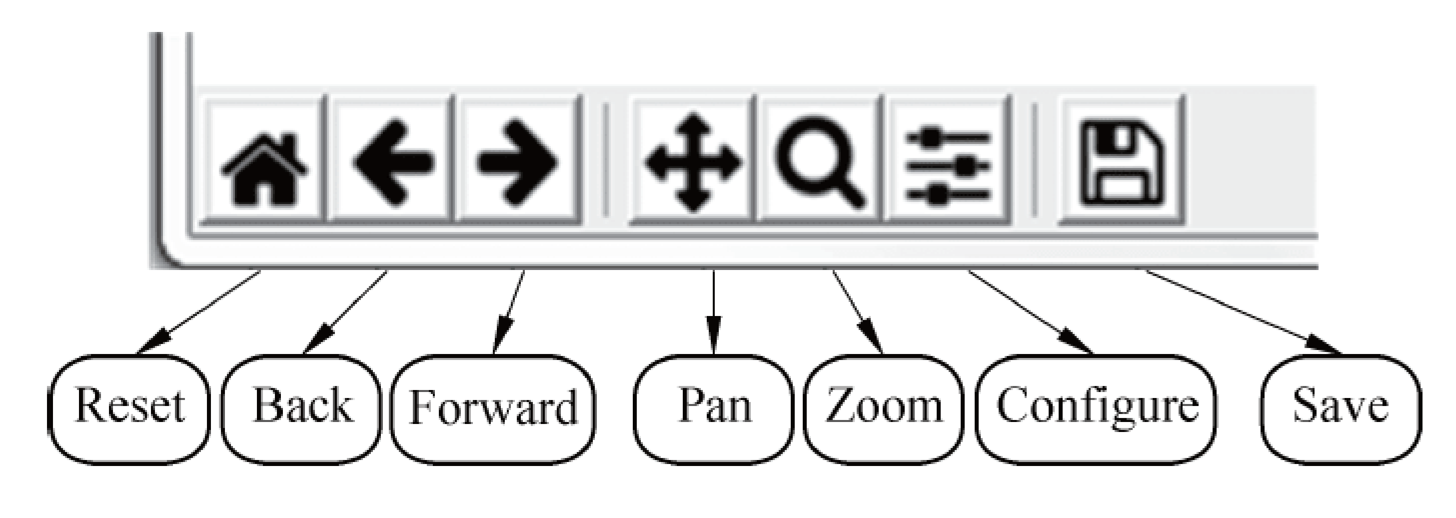


图 6.3　交互界面功能键

这些功能键依次介绍如下。

（1）Reset：返回初始视图，这是相对于改变图样式之后而言的。

（2）Back/Forward：这两个按钮有点像浏览器中的后退和前进功能，可以单击它们以实现不同操作效果的重现。

（3）Pan：平移操作，单击之后鼠标光标会改变并且可以单击并拖曳显示出的图。另外按住右键拖曳可以旋转，缩放。

（4）Zoom：放大操作，左键单击选定区域可以将选出区域内容放大，右键单击选定区域可以将选出区域内容缩小。

（5）Configure：对图形和绘图配置，单击后会弹出如图 6.4 所示的界面，通过调整滑块来改变间距，其中 wspace 和 hspace 是用于调整界面中有多个子图时子图之间的水平间距和垂直间距，当没有子图时调整不会有变化。界面中的 Reset 按钮会重置所有设置。

（6）Save：保存图形，单击后会弹出保存界面。

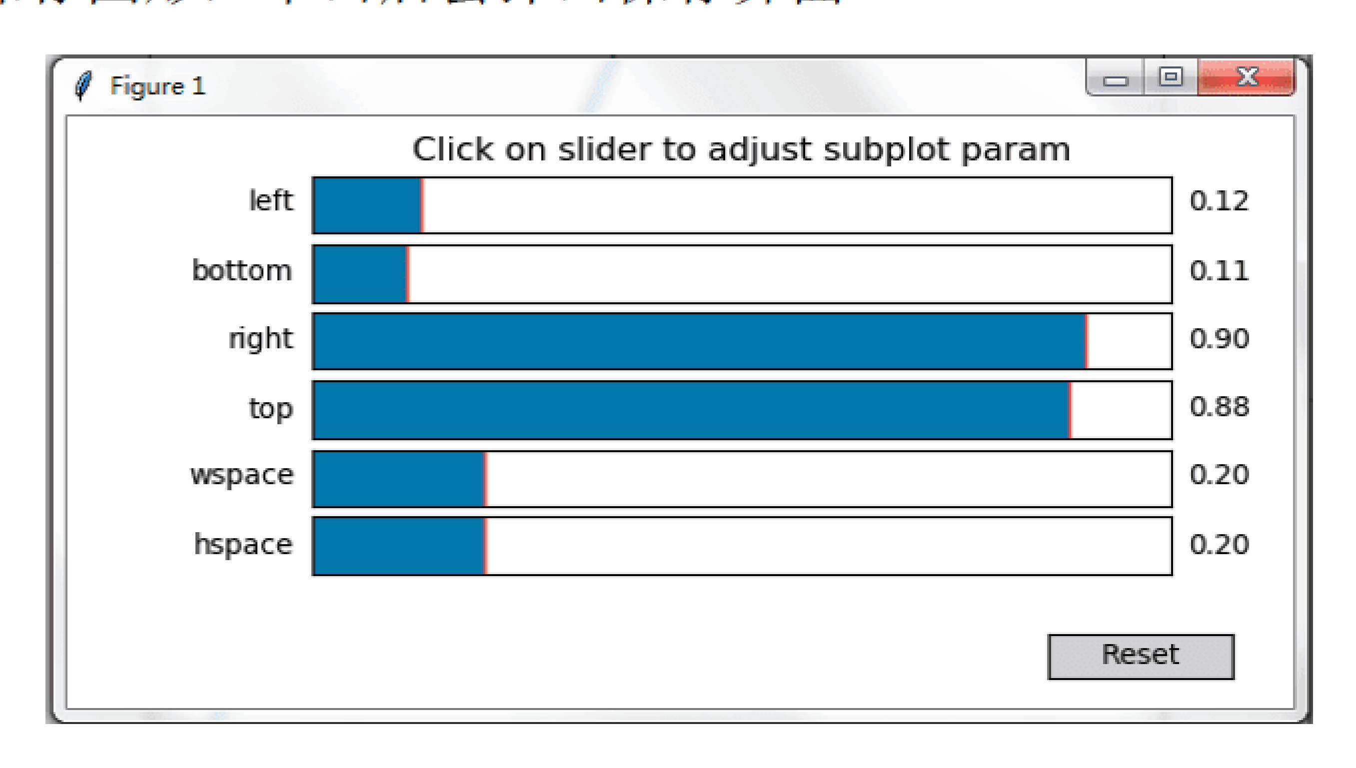


图 6.4　Configure 配置界面

6.2　pandas 绘图基础

pandas 是一个功能强大的 Python 数据分析模块，它有着对齐功能的数据结构，丰富的数学运算以及灵活的缺失书籍处理功能。在使用之前先要安装模块，可以使用语句 pip install pandas 安装，另外也可以使用上一节介绍的在 PyCharm 中的安装方法安装 pandas 模块。

在本节中将重点放在使用 pandas 模块和 matplotlib 模块结合画图上，我们将 pandas 处理后的数据再使用 matplotlib 模块画出。接下来先来看看下面这段程序。

```
1:    import numpy as np
2:    import pandas as pd
3:    import matplotlib.pyplot as plt
4:    data = pd.Series(np.random.rand(1000))
5:    data.plot()
6:    plt.show()
```

这段代码创建了 1000 个随机数并用于显示，前 3 行是程序中用到的模块的引入，其中第 2、3 行是之前介绍的 pandas 模块和 matplotlib 模块。第 1 行中的 numpy 是一个高性能科学计算和数据分析模块，它是 pandas 模块使用的基础。其中 np.random.rand(1000)是产生 1000 个随机数，Series 是由数据和数据对应的索引组成的，类似于列表和字典的结合，是有序对象并存在变量 data 中。在上一节中，显示图形需要使用 plt.plot()函数，在本段代码中使用的是 data.plot()显示，这是因为 pandas 中支持直接使用 plot()方法显示关于数据的图形，*x* 轴的数据为索引值，*y* 轴为之前产生的随机数。最后使用 show()方法将图形调出。上述代码段画出的图形如图 6.5 所示。

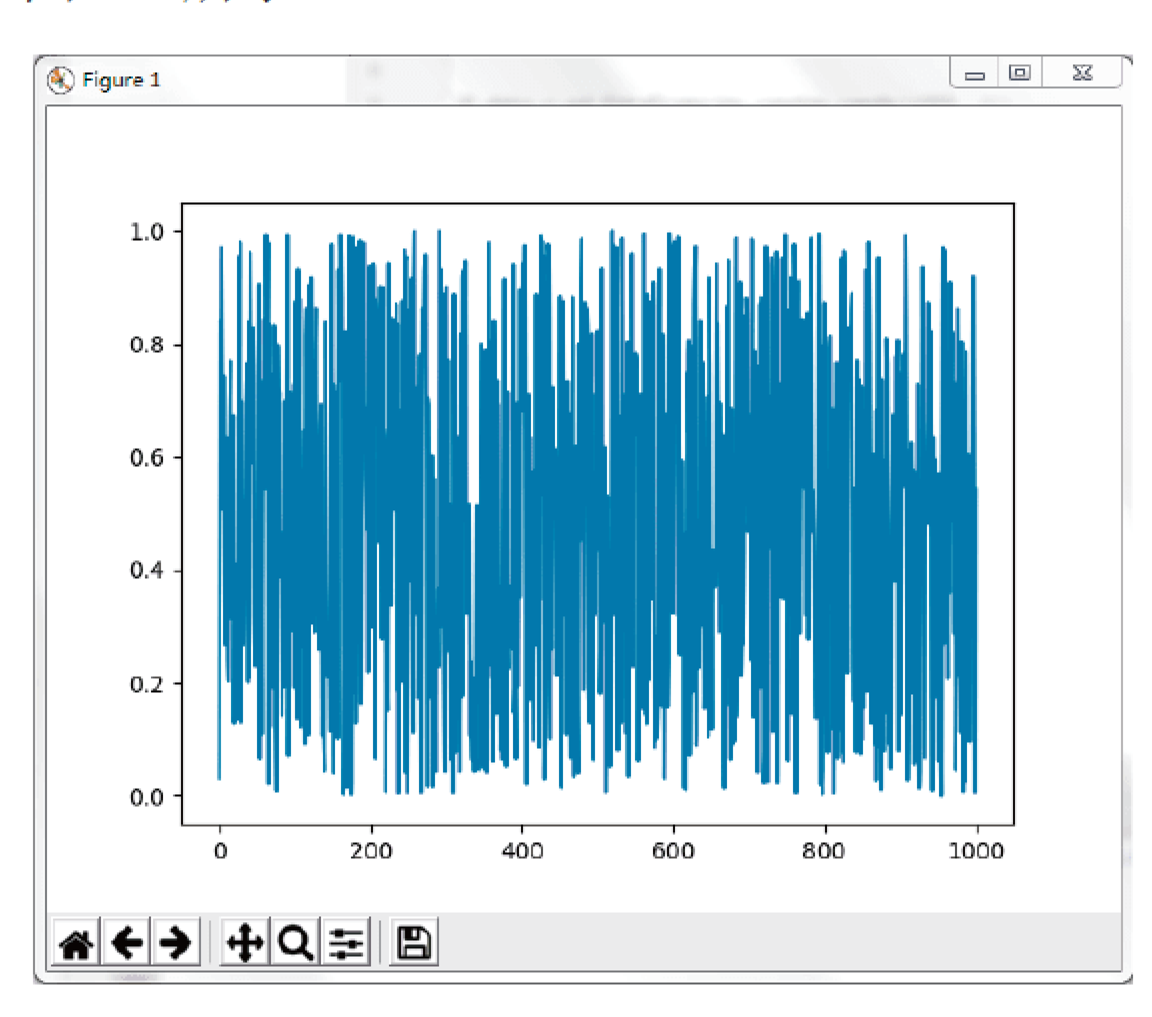


图 6.5　pandas 随机数生成的图

6.3　基本图形的绘制

本节将使用小程序的方式直观地列出不同图形的绘制方式，并分析程序中的关键部分，让大家快速学会画图。

6.3.1　折线图以及标题和标签

在前文中介绍的图都是折线图，在这里我们详细介绍一下将标题和标签添加到显示的图形上。具体地说，matplotlib 中我们使用 plt.plot()将绘图的数据作为参数传入，以此来绘制一个折线图，完成之后使用 plt.show()显示，先来看看程序 6.1。

程序 6.1　折线图：

```
import matplotlib.pyplot as plt

x1 = [1, 3, 4]
y1 = [2, 4, 1]
x2 = [1, 3, 4]
y2 = [4, 1, 5]

plt.plot(x1, y1, label = 'Line1')
plt.plot(x2, y2, label = 'Line2')

plt.xlabel('X-axis')
plt.ylabel("Y-axis")
plt.title('test Graph')
plt.legend()
plt.show()
```

输出图形，如图 6.6 所示：

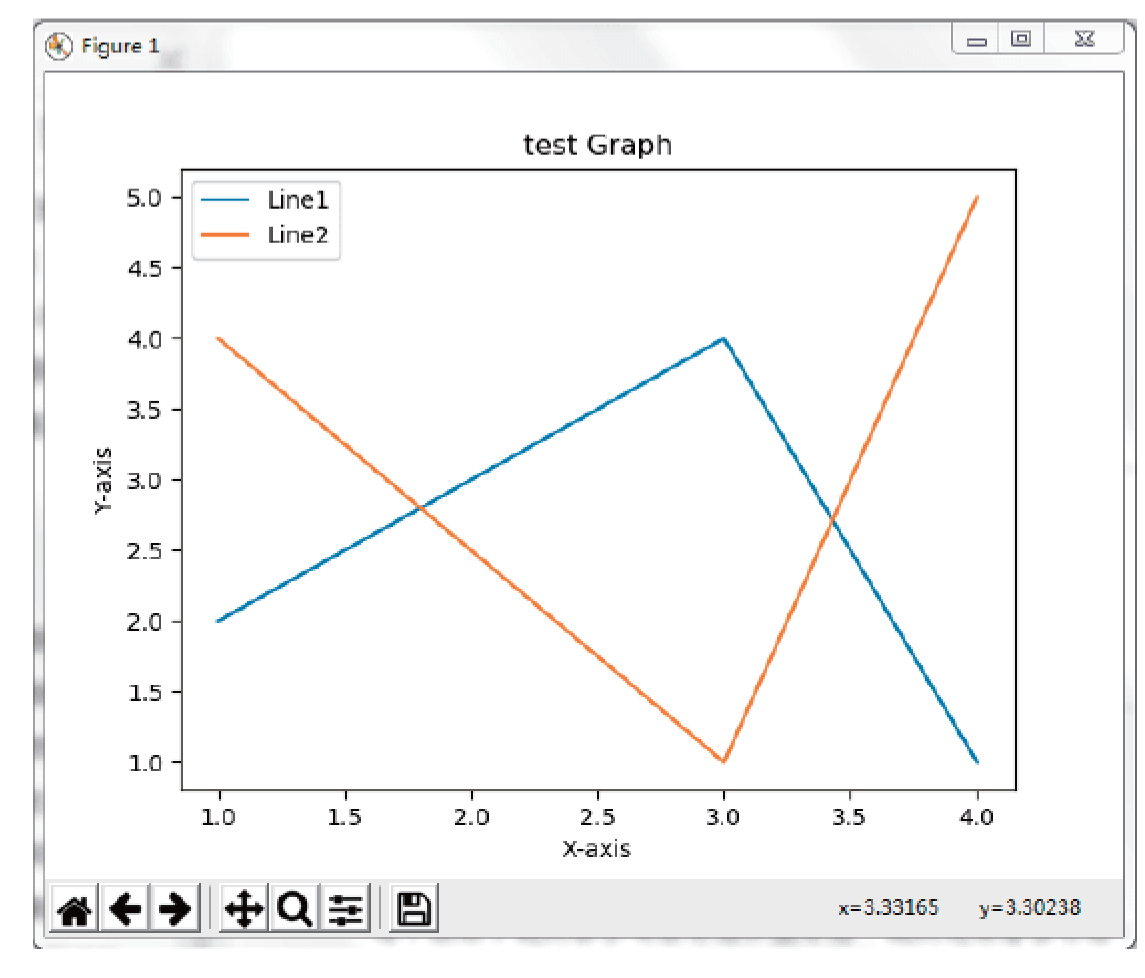


图 6.6　程序 6.1 输出结果

分析：

在该程序中使用数据存入列表中并传入 plot()中用于绘图，同时传入 label 关键字用于显示时标注折线名，还可传入 color 关键字用于指定直线颜色，例如，color = g 可以将直线的颜色指定为绿色，color 的值也支持 16 进制的颜色代码，如#191919。

设置完直线后开始标注坐标轴信息，程序的第 11、12 行用于设置坐标名称，第 13 行用于设置图的标题名。之后使用 plt.legend()生成默认图例，最后使用 plt.show()显示图形。

该程序在同一坐标系画出了两条直线，这在之后的编程中是很实用的，因为我们经常要将数据对比，而图形又是一种很直观的表现形式。将两条直线放在同一坐标系中并不是难事，只需要使用 plot 两次创建直线，同时使用 label 指出用于区分直线的名称即可。

6.3.2　条形图

之前所涉及的全是折线图的画法，接下来看看条形图的画法，如程序 6.2 所示。

程序 6.2　条形图：

```
import matplotlib.pyplot as plt

x1 = [1, 3, 5, 7, 9, 11]
y1 = [4, 1, 5, 7, 10, 5]

x2 = [2, 4, 6, 8, 10, 11]
y2 = [1, 3, 9, 2, 11, 2]

plt.bar(x1, y1, label = "test1")
plt.bar(x2, y2, label = "test2")

plt.xlabel('X-axis')
plt.ylabel("Y-axis")
plt.title('test Graph')
plt.legend()
plt.show()
```

输出图形，如图 6.7 所示：

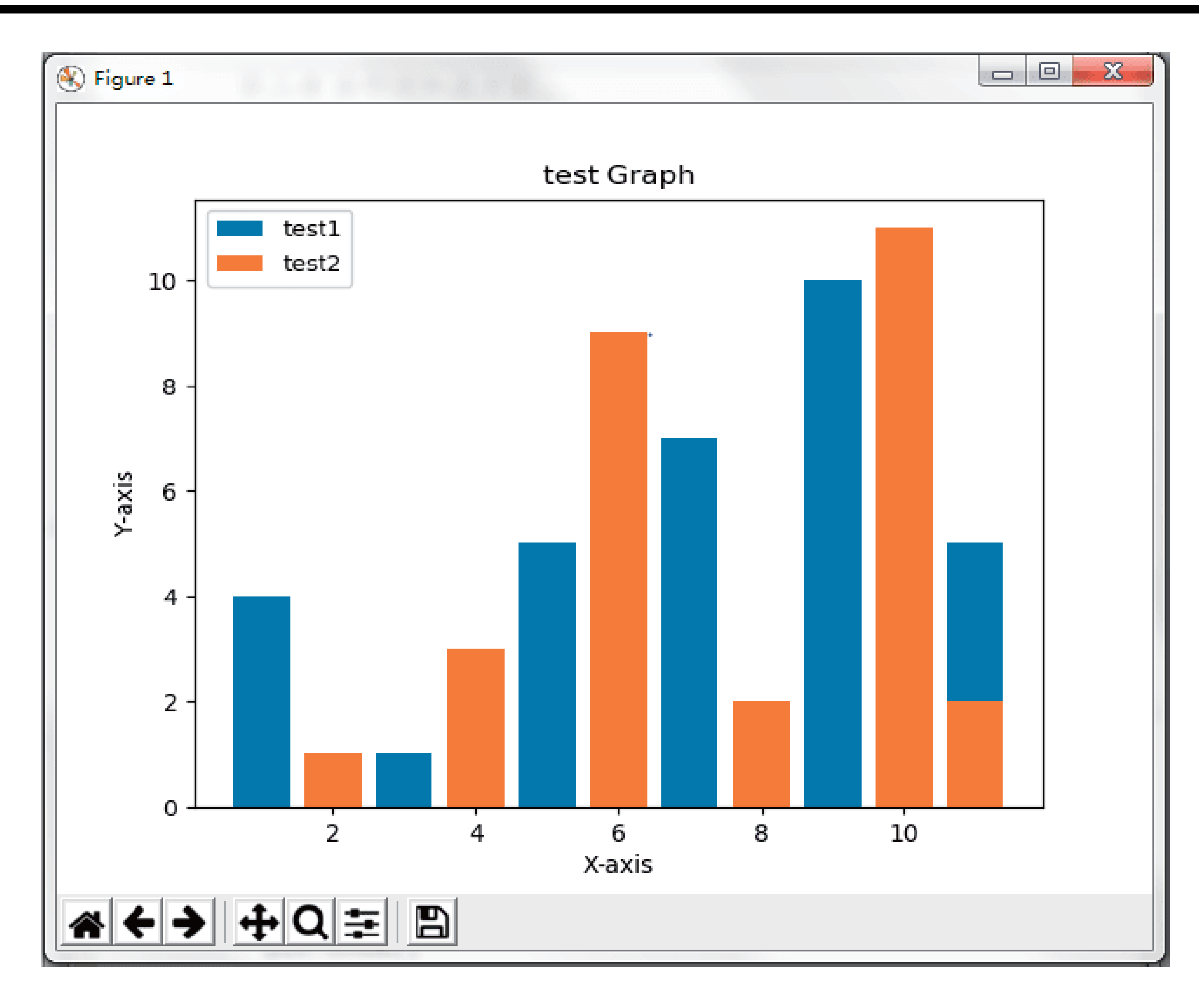


图 6.7　程序 6.2 输出结果

分析：

将程序 6.2 同程序 6.1 对比会发现其实代码结构是相同的，并且语句内容也很相似，除了画图的数据以及图的标注部分有些差异之外，最大的差别在于条形图使用的是 plt.bar()来创建图形（程序的第 9 行），折线图是用 plt.plot()来创建图形。从图形的数据源来说，当 x 轴有重合时图形会叠在一起，正如图 6.7 中 x 轴值为 11 的图形所示。

6.3.3　直方图

直方图同条形图非常相似，但是它们表述的内容却不太一样，条形图从数据源的角度来说还是有些像坐标数据，但是直方图的数据源有点偏向于统计学，接下来看一个简单的程序 6.3。

程序 6.3　直方图：

```
import matplotlib.pyplot as plt

data = [5, 22, 10, 44, 55, 99, 87, 86, 45, 22, 6, 33, 87, 98, 15, 25, 77, 89, 31, 32, 71, 83, 82, 52 ]
scalar = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.hist(data, scalar, histtype='bar', rwidth=0.5)

plt.xlabel('X-axis')
```

```
8:   plt.ylabel("Y-axis")
9:   plt.title('test Graph')
10:  plt.legend()
11:  plt.show()
```

输出图形，如图 6.8 所示：

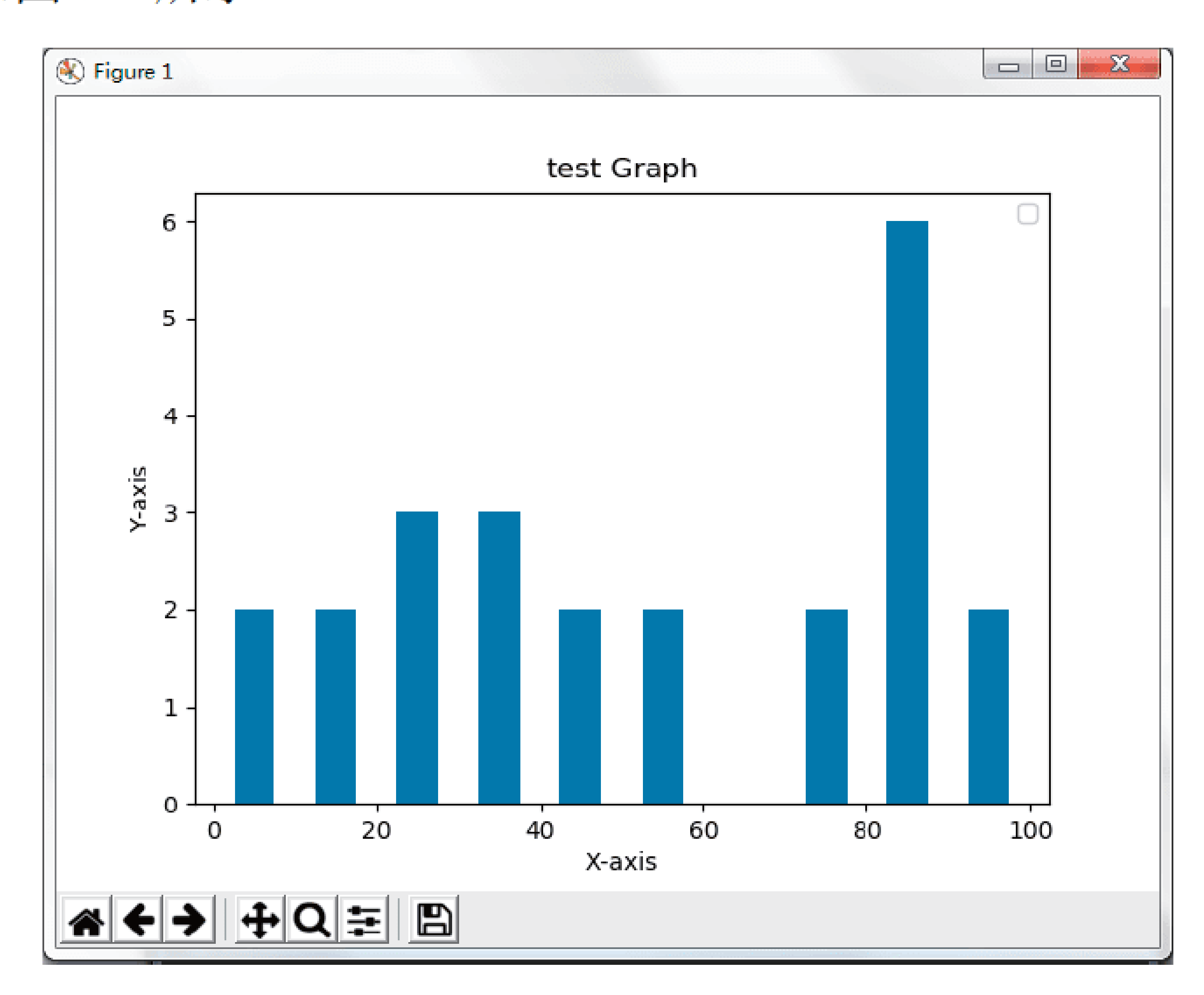


图 6.8　程序 6.3 输出结果

分析：

从程序 6.3 可以看出创建图形处的代码已经和之前不同，对于直方图的数据需要先建立一个数据列表，接着是一个量度的列表，之后直方图会自动地计算出数据列表中数据在不同量度区间的数据个数。如图 6.8 所示，在 60～70 处没有图形，这意味着数据列表中没有处于 60～70 之间的数据。

再来看看程序的第 5 行 plt.hist(data, scalar, histtype='bar', rwidth=0.5)，这条语句中前两个参数表示上文提到的数据列表和量度列表，参数 histtype='bar'表示直方图显示类型，histtype 的值也可以是 barstacked、step 或 stepfilled，具体效果不在这里展示了，rwidth 用于指定显示的条形图中的每个条的宽度。同时，也可以添加 facecolor 直方图颜色参数或是添加 alpha 透明度参数等。

程序中剩余部分同程序 6.2 一样，这里不再重复介绍。

6.3.4　散点图

散点图在科学分析中是非常常见的，它主要用于分析对象相关性的体现。在具体的程序中，散点图的实现方式非常简单，具体程序如程序 6.4 所示。

程序 6.4　散点图：

```
import matplotlib.pyplot as plt
x = [1, 3, 5, 7, 9, 11]
y = [4, 1, 5, 7, 10, 5]

plt.scatter(x, y, label="scatter", s=25, marker="o")

plt.xlabel('X-axis')
plt.ylabel("Y-axis")
plt.title('test Graph')
plt.legend()
plt.show()
```

输出图形，如图 6.9 所示：

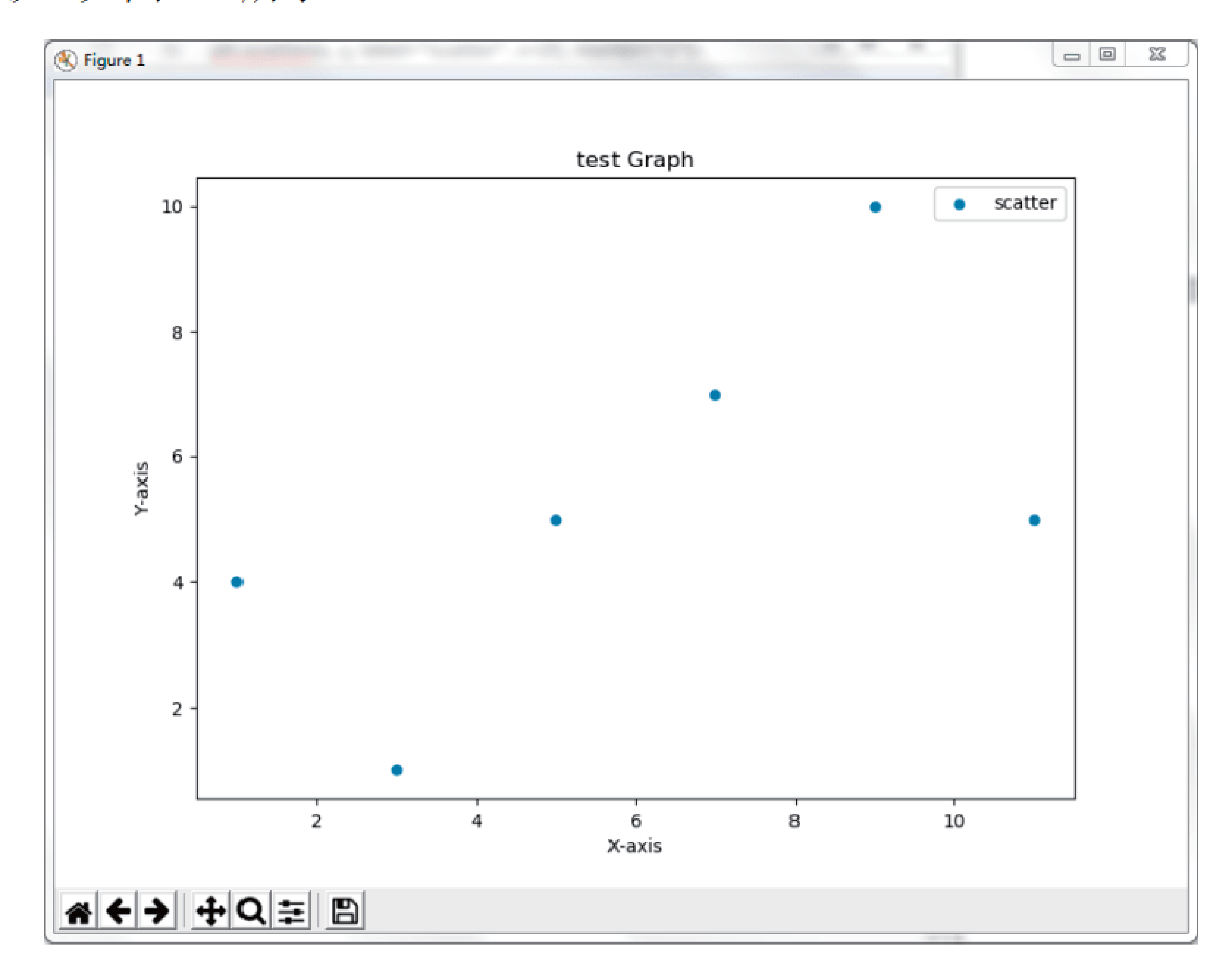


图 6.9　程序 6.4 输出结果

分析：

程序 6.4 和程序 6.1、程序 6.2 非常像，其中最大的不同在于该程序的第 5 行是用于创建散点图的语句。在程序的第 5 行中，使用 plt.scatter 指定创建的是散点图，同之前一样，语句括号中的内容作为参数传入，用于指定图形的具体形式。同之前程序一样，变量 x 和 y 分别为数据源传入，label 为用于表示数据的标签，s 为图中用于显示散点的大小，marker 为散点在图中的表示形式，它的值"o"表示使用圆的形式在图中显示散点，还有些其他的表示方式如表 6.1 所示。另外还有些像是 color 等常用的参数使用形式同之前讲的相同。

表 6.1　常用的 marker 及其对应表现形式

marker 的值	描　　述
","	使用像素（正方形）的形式表示散点
"v"	使用下三角的形式表示散点
"x"	使用符号×的形式表示散点
"d"	使用菱形的形式表示散点
"*"	使用五角星的形式表示散点

6.3.5　饼图

饼图同之前介绍过的图形都不一样，它用于显示部分内容对于整体而言的所占比例情况。对于所占比，我们只需要提供数据源的数据，matplotlib 模块会自动计算出每部分的所占比，例如，下面的程序计算某位同学考试各科成绩在总成绩中的占比。具体程序如程序 6.5 所示。

程序 6.5：

```
import matplotlib.pyplot as plt
Score = [95, 89, 65, 90]
Subject = ['Math', 'Chinese', 'English', 'Synthetical']
cols = ['c', 'm', 'r', 'b']

plt.pie(
    Score,
```

```
8:          labels=Subject,
9:          colors=cols,
10:         startangle=90,
11:         shadow=False,
12:         explode=(0.1, 0, 0, 0),
13:         autopct='%1.1f%%')
14:     plt.title("test Graph")
15:     plt.show()
```

输出图形，如图 6.10 所示：

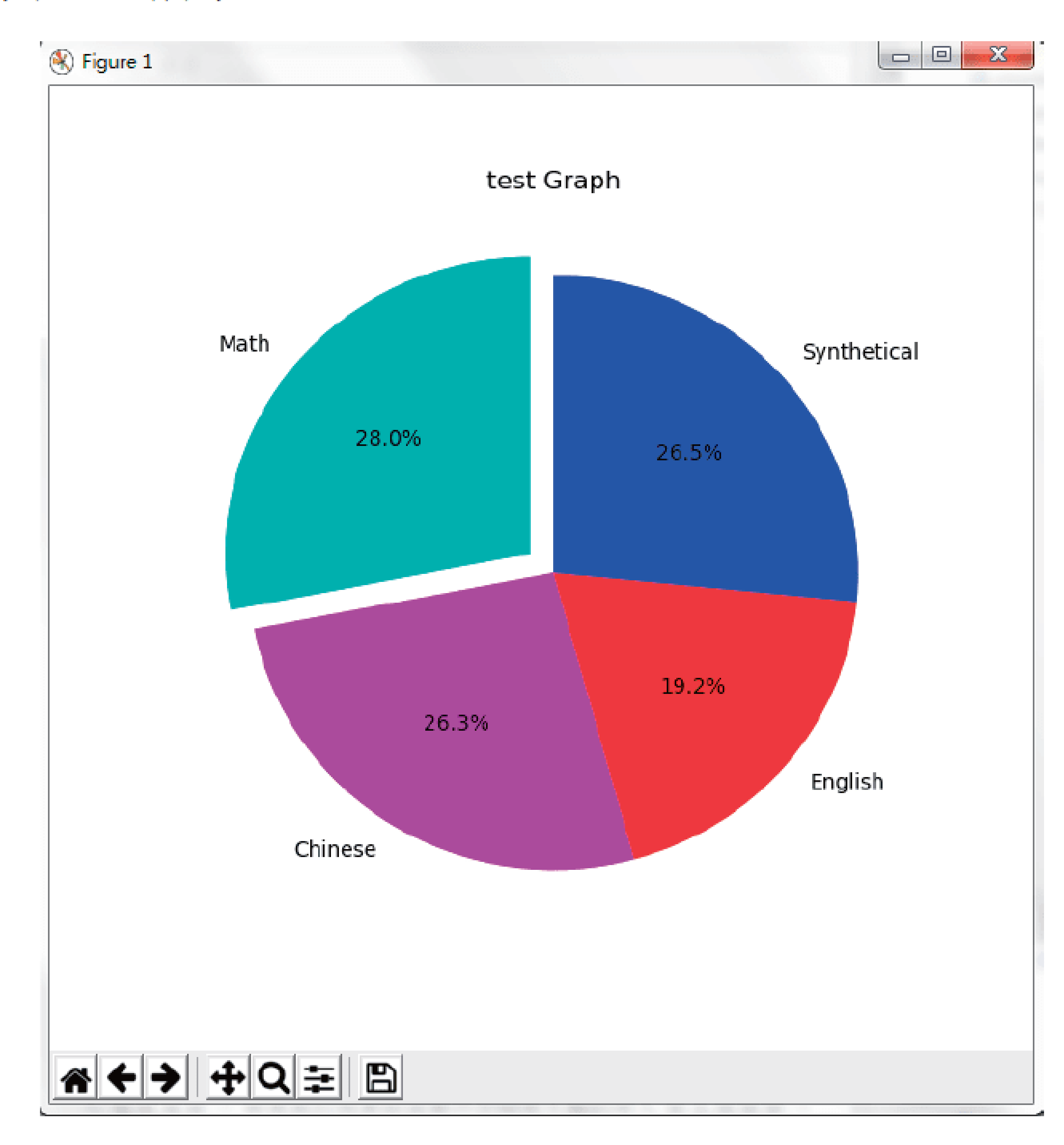


图 6.10　程序 6.5 输出结果

分析：

程序 6.5 中假设学科分为 4 门，分别是 Math（数学）、Chinese（语文）、English（英语）和 Synthetical（综合）。这 4 门课程的标签存在列表 Subject 中，对应的分数存在列表 Score 中。我们从程序的第 6 行切入，首先提供了画图需要的分数数据和标签数据，接着参数 colors 是按照顺序对应的方式设置了各个部分的颜色，参数 startangle 设置了第 1 个部分即程序中 Math 部分在饼图中的起始角度，程序中设置的是 90°，这个体现在输出

图形中 Math 部分的右边界为垂直角度。通过设置 startangle 的角度可以实现调整整个饼图的数据角度。参数 shadow 表示是否为带有阴影的饼图，参数 explode 用于指定的部分拉出，例如，程序的第 12 行传入 explode=(0.1, 0, 0, 0)将第 1 部分（分数列表中第 1 项对应的标签）Math 拉出，而如果想要把 English 拉出，将 explode 赋值为(0, 0, 0.1, 0)即可。最后一个参数 autopct 是用于选择将百分比放到图中各个部分上，并将其值精确到小数点后一位。

6.3.6　图像注释

接下来使用之前讲过的 pandas 中的数据产生散点图并对程序生成的图像进行注释。先来看程序 6.6。

程序 6.6　图像注释：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.DataFrame(np.random.randn(1000, 2), columns=list("AB"))
print(data.head())
data.plot.scatter(x='A', y='B')

plt.text(2, 2, 'Simple A')
plt.annotate('boundary', xy=(-3,-2), xytext=(-2.5,-2.5), arrowprops=dict(facecolor='red'))
plt.grid(True)
plt.show()
```

输出：

```
          X          Y
0  0.482817 -2.202961
1  0.297005 -0.452913
2  0.434574 -1.007296
3  0.672244   0.687188
4  0.342619 -1.055706
```

输出图像，如图 6.11 所示：

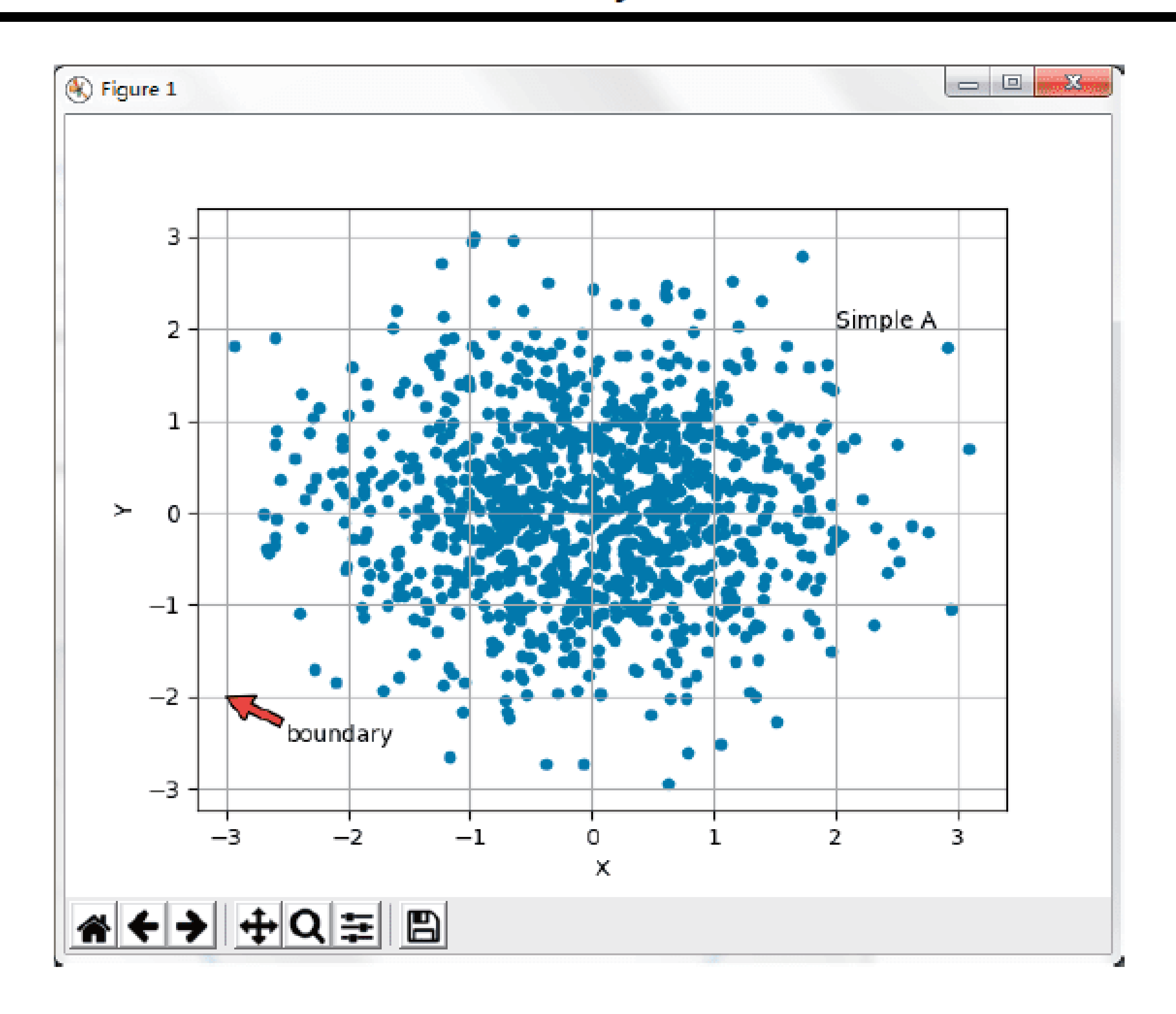


图 6.11　程序 6.6 输出结果

分析：

程序的第 5 行使用 pandas 模块创建了 1000 个 2 列正态分布的随机数，这些数的前 5 个可以在输出中看到。程序的第 7 行根据之前设置的列名 x、y 调用列中的数据用于显示，这些步骤基本和之前介绍的 pandas 模块数据创建折线图相似。

接下来看看如何给生成的图添加注释，程序的第 9 行使用 plt.text()给图像添加一条文本注释，其中前两个参数表示注释在图像起始位置的坐标，第 3 个参数用字符串的形式表示注释的内容。程序的第 10 行使用 plt.annotate()给图像添加一个箭头加文本用于指定图像中具体内容的注释，其中第 1 个参数是字符串表示注释的内容，第 2 个参数是箭头的坐标，第 3 个参数是注释内容的坐标，最后一个参数用于指定箭头颜色。

完成这些设置后，使用 plt.grid(True)将图像的网格显示出来，完成这些后使用 show()方法显示图像。

6.3.7　子图

子图是在一个输出图像中显示多个图形，可以将其理解为分域，每个区域都有独自的执行代码用于创建当前区域的图。接下来看看具体创建子图的程序 6.7。

程序 6.7：

```
import matplotlib.pyplot as plt

```

```
3:   fig = plt.figure()
4:   x = [1, 3, 5, 7, 9, 11, 15, 17]
5:   y = [4, 1, 5, 7, 10, 5, 0, 20]
6:
7:   ax1 = fig.add_subplot(2, 2, 1)
8:   ax2 = fig.add_subplot(2, 2, 2)
9:   ax3 = fig.add_subplot(2, 1, 2)
10:
11:  ax1.scatter(x, y)
12:  ax1.set_title('scatter graph')
13:  ax2.plot(x, y)
14:  ax2.set_title('plot graph')
15:
16:  ax3.bar(x, y)
17:  ax3.set_title('bar graph')
18:  ax3.set_xlabel('X')
19:  ax3.set_ylabel('Y')
20:  ax3.annotate('first one', xy=(1,1), xytext=(2,2), arrowprops=dict(facecolor='r'))
21:  plt.show()
```

输出图像，如图 6.12 所示：

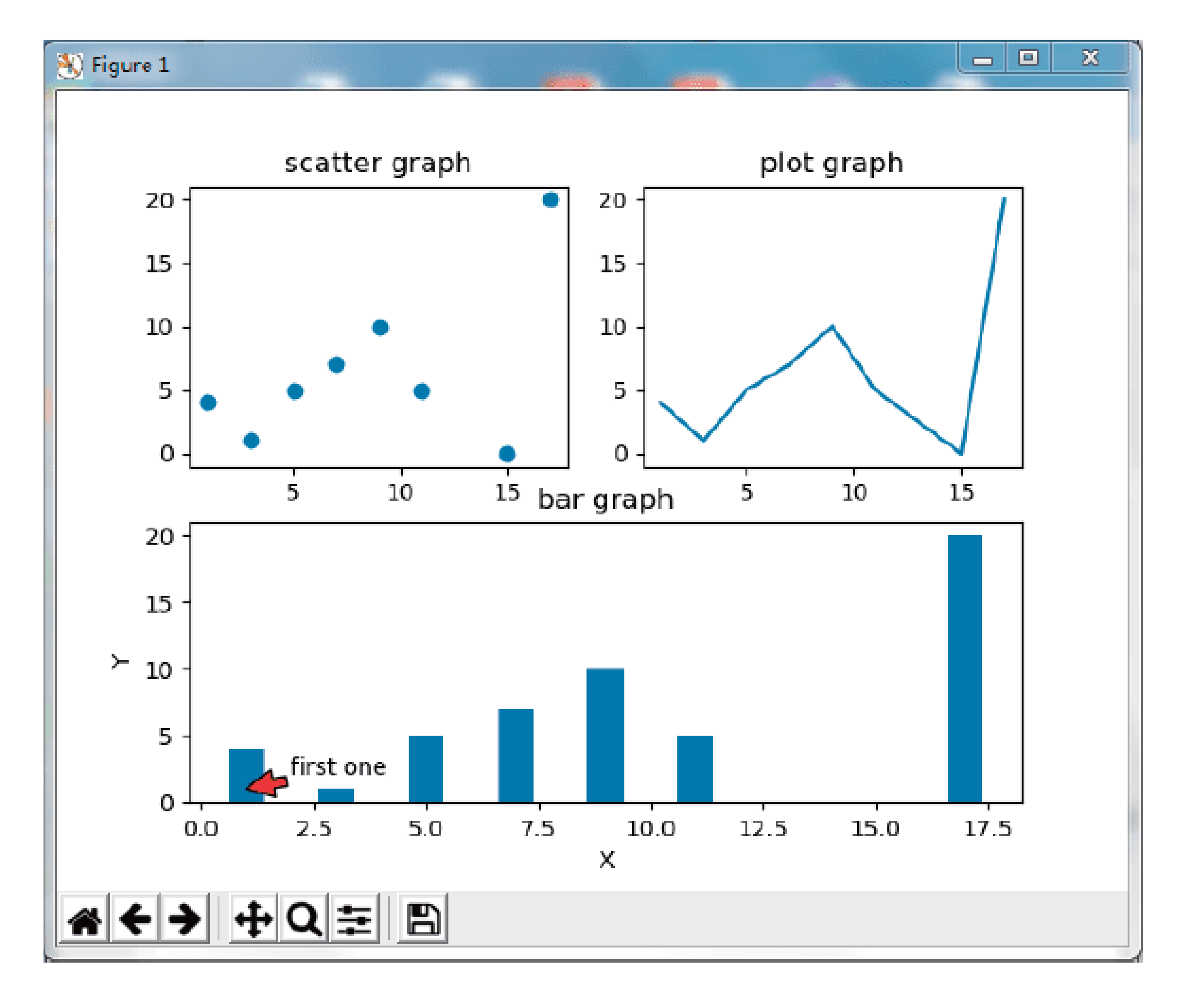


图 6.12　程序 6.7 输出结果

分析：

程序的第 3 行使用 plt.figure()得到一个绘图对象并赋值给变量 fig，后续的操作都作用

在 fig 上。第 7～9 行使用 fig.add_subplot 对图像分域，并在区域上添加对应的子图。它的参数分别是行数、列数以及与规则对应的子图编号。下面我们来谈谈这个对应规则，子图在整个图像的位置是由参数的 3 个数字决定的，参数 2, 2, 1 表示的是 2 行 2 列（2×2）的第 1 个位置，参数 2, 2, 2 表示的是 2 行 2 列的第 2 个位置，参数 2, 1, 2 表示的是 2 行 1 列（2×1）的第 2 个位置，具体的子图分布如图 6.13 所示。

图 6.13（a）表示 2 行 2 列的子图编号分布，图 6.13（b）表示 2 行 1 列的子图编号分布。例如，参数 2, 2, 1 对应的就是图 6.13（a）中编号为 1 的位置，参数 2, 1, 2 为对应图 6.13（b）编号为 2 的位置。

设置完成子图之后开始对各个子图单独设置，第 11～14 行使用之前的数据创建了散点图和折线图，这些操作和上文提到的差不多，要注意的是这里设置标题要使用 set_title。

程序的第 16～20 行创建并设置了第 3 个子图，同样，设置 x，y 轴的标签名也要加上 set_。虽然之前对子图设置都要加上 set_，但是添加注释的方法（程序的第 20 行）则和之前介绍的方法一样。

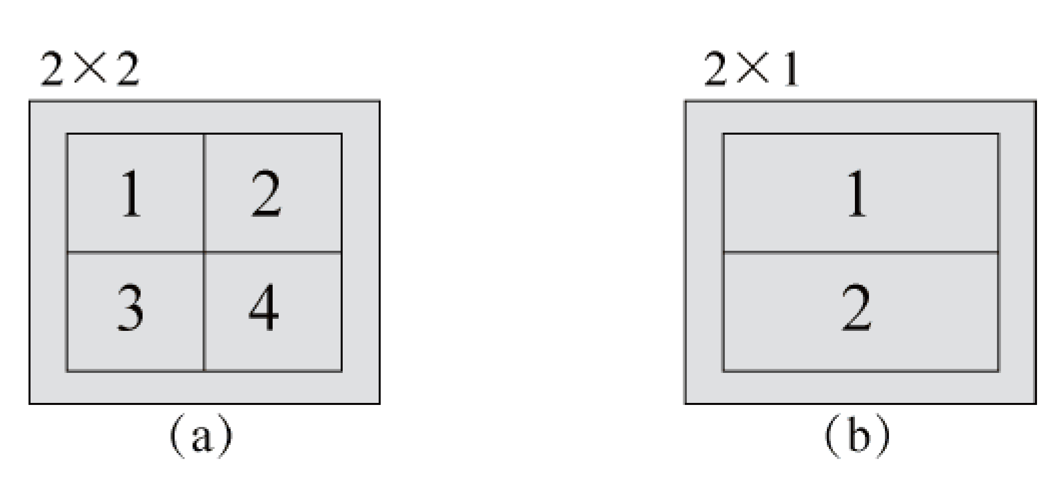


图 6.13　子图分布

6.3.8　3D 散点图

matplotlib 模块也支持 3D 图形，相对于 2D 图形，3D 允许我们添加第三组变量。如果说 2D 图形用于表示两个值之间的关系，那么 3D 图形用于表示三个值之间的关系，可以用于更为复杂变量之间的直观表示。3D 散点图的实现如程序 6.8 所示。

程序 6.8　3D 散点图：

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(1, 1, 1, projection='3d')

x1 = [1,2,3,4,5,6,7,8,9,10]
```

```
8:    y1 = [5,6,7,8,2,5,6,3,7,2]
9:    z1= [-1,-2,-6,-3,-2,-7,-3,-3,-7,-2]
10:   x2 = [-1,-3,-5,-7,-9,-11,-12,-15,-16,-17]
11:   y2 = [4,1,5,7,10,5,12,8,9,13]
12:   z2 = [1,2,6,3,2,7,3,3,7,2]
13:
14:   ax1.scatter(x1, y1, z1, color='b', marker='o')
15:   ax1.scatter(x2, y2, z2, color='r', marker='x')
16:   ax1.set_xlabel('x axis')
17:   ax1.set_ylabel('y axis')
18:   ax1.set_zlabel('z axis')
19:   plt.show()
```

输出图像，如图 6.14 所示：

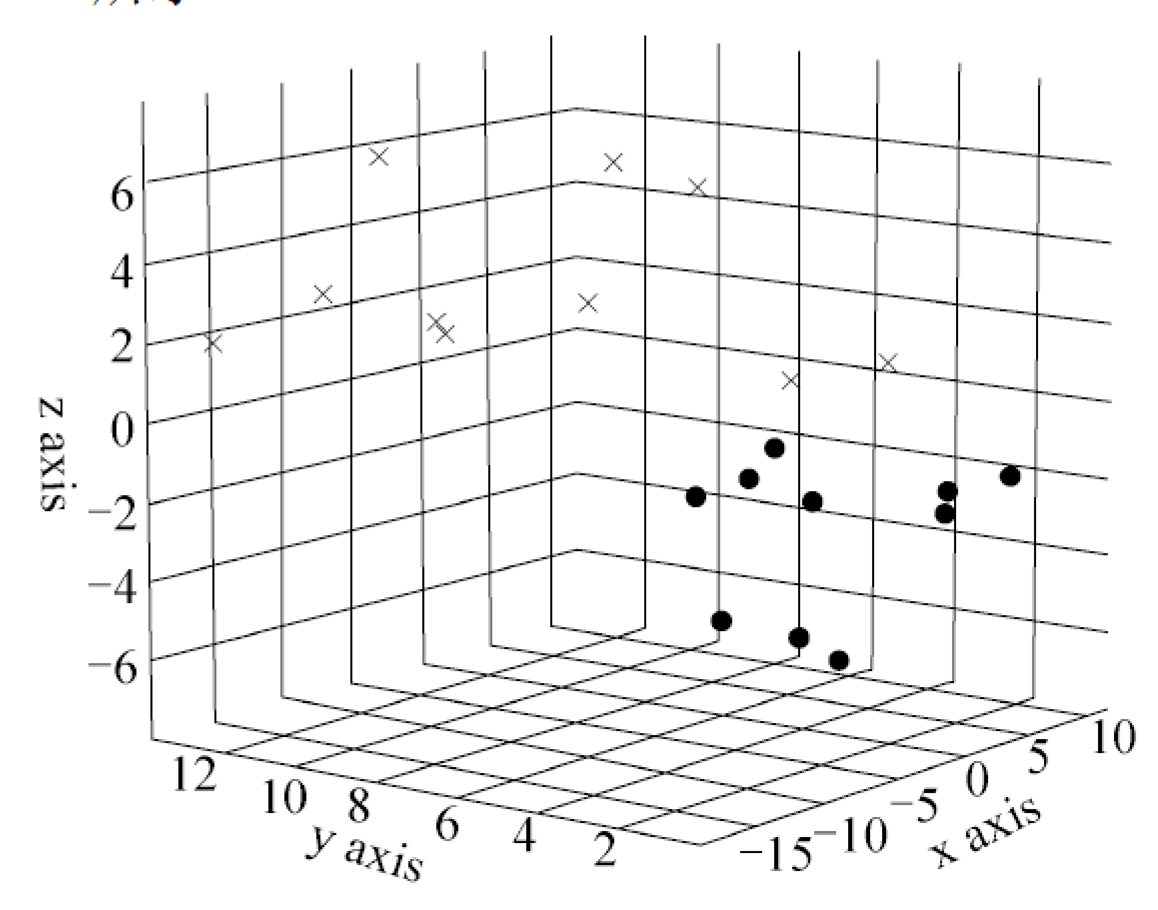


图 6.14　程序 6.8 输出结果

分析：

程序 6.8 总体来说较为简单，有许多语法在之前都提到过。首先第 1 行用于将画 3D 图形的模块引入，程序的第 5 行将 ax1 设置为一个单位的子图，因为参数为 1，1，1，因此整个图中只有一个子图。接下来设置好参数之后使用 ax1.scatter 和 ax2.scatter 建立散点图并对其进行设置，从程序生成的图像可以看出两类散点图已经完成，同时，我们还可以在交互界面对其拖曳旋转查看图像的不同角度。

6.3.9　3D 条形图

介绍完 3D 散点图后，我们来看看 3D 条形图。不同于散点图，3D 条形图不仅要设置

坐标点，还要设置条形的宽度、高度和深度，具体程序如程序 6.9 所示。

程序 6.9　3D 条形图：

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111, projection='3d')

x = [1, 2, 3]
y = [1, 2, 3]
z = [2, 2, 2]

dx = [1, 2, 3]
dy = [1, 1, 1]
dz = [1, 1, 3]

ax1.bar3d(x, y, z, dx, dy, dz)
ax1.set_xlabel('x axis')
ax1.set_ylabel('y axis')
ax1.set_zlabel('z axis')

plt.show()
```

输出图像，如图 6.15 所示：

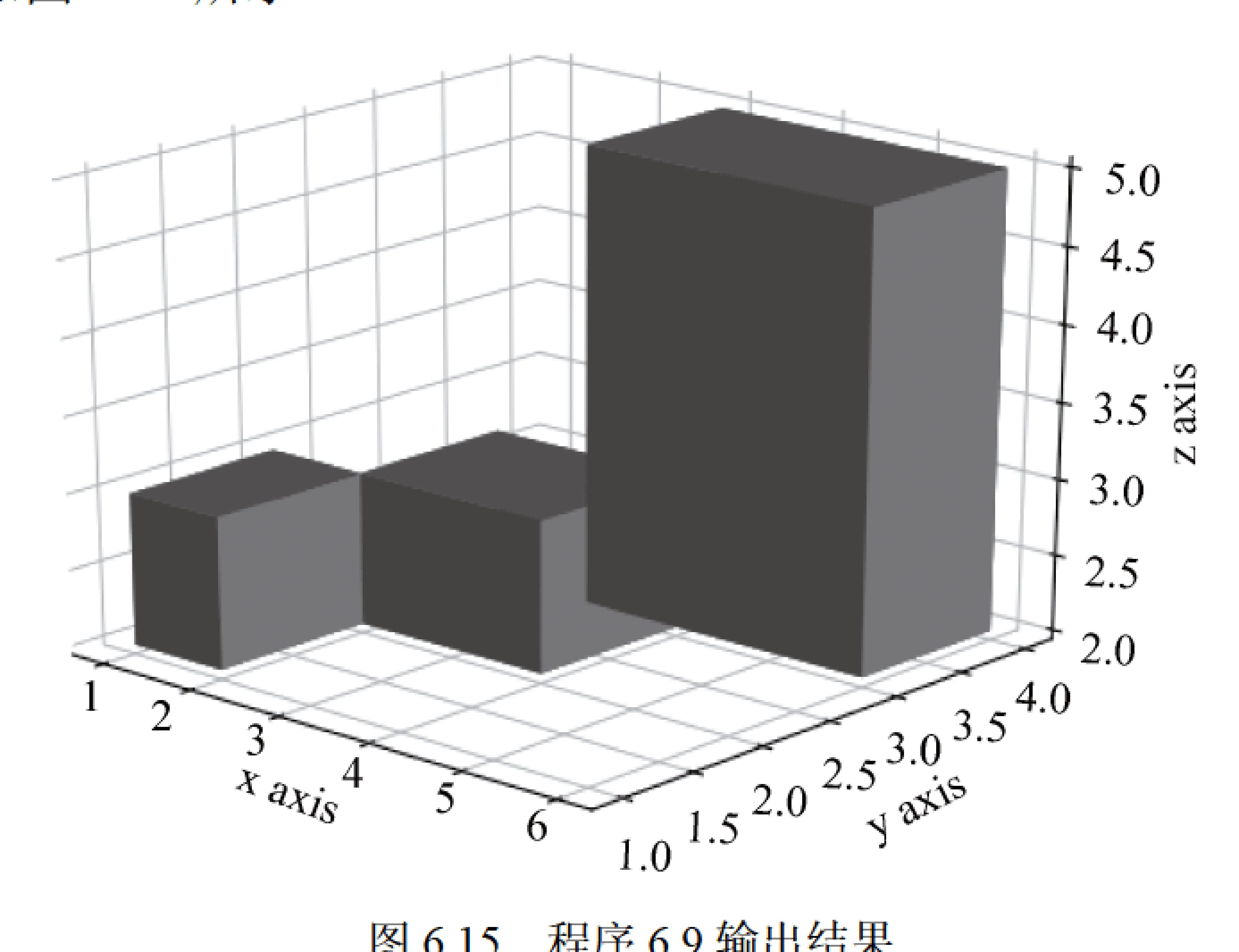


图 6.15　程序 6.9 输出结果

分析：

程序 6.9 中的大部分语句和程序 6.8 相同，最大的不同之处在于程序的第 15 行使用 ax1.bar3d(x, y, z, dx, dy, dz)来创建了一个条形图。其中 x,y,z 表示条形的起始点坐标，dx 表示宽度即从起始点开始沿 *x* 轴延伸的长度，dy 表示深度即从起始点开始沿 *y* 轴延伸的长度，dz 表示高度即从起始点开始沿 *z* 轴延伸的长度。图 6.16 以程序中的第三个条形为例，详细介绍了这几个参数在图中所指。完成 3D 条形图的创建之后，依照子图的标签设置方式给图像添加标注，最后显示图像。

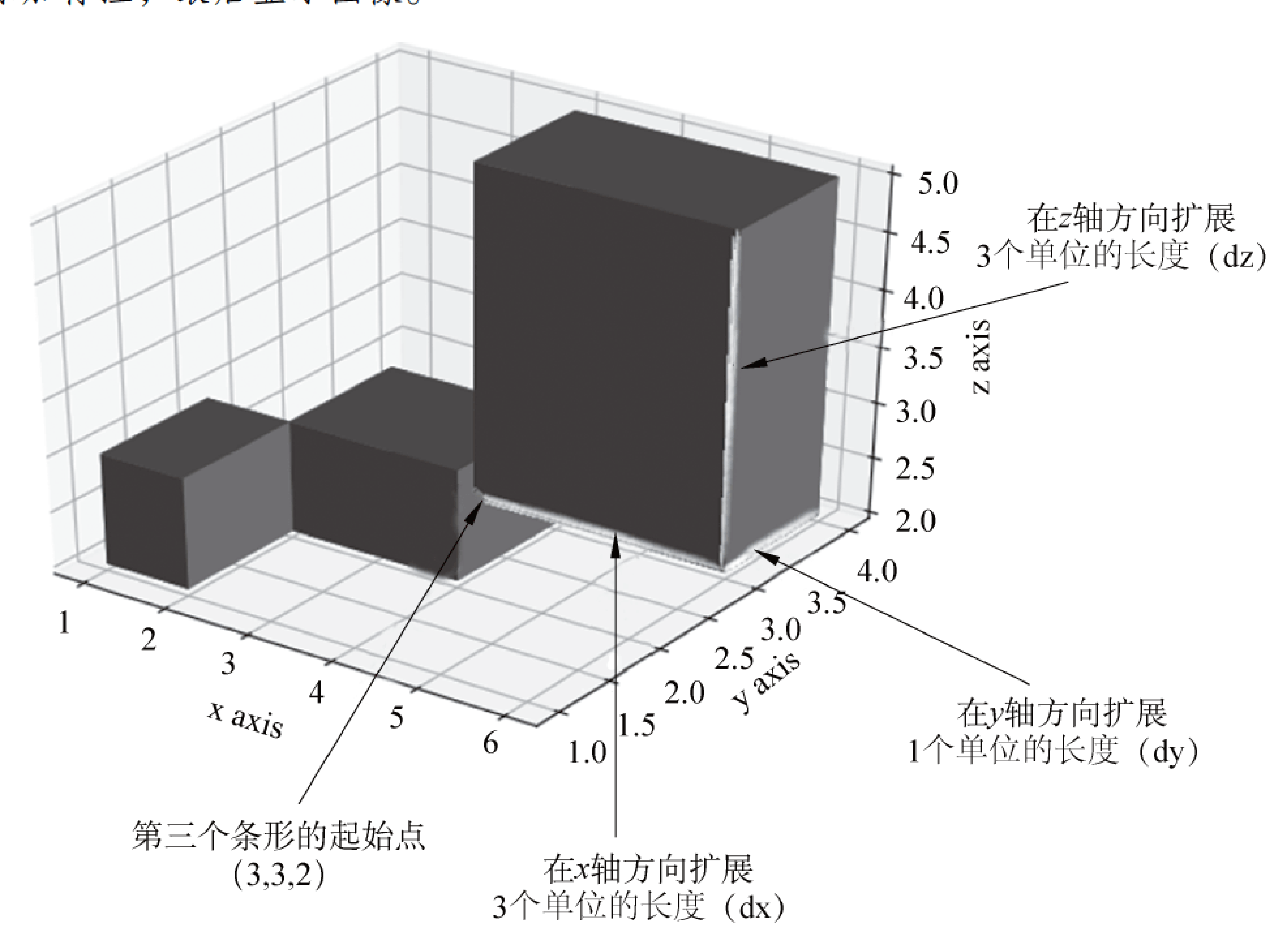


图 6.16　条形图第三个条形参数的具体所指

6.4　绘制正弦交变电流图像

学习完上述的绘图内容后，我们来处理一个在物理课上学习过的内容，输出周期为 0.02s 的我国生活用 220V 交流电的图像，具体程序如程序 6.10 所示。

程序 6.10：

```
1:  import numpy as np
2:  import matplotlib.pyplot as plt
3:
4:  x = np.arange(0.0, 5.0, 0.001)
5:  y = 220 * np.sin(100*np.pi*x)
6:
7:  plt.plot(x, y)
8:  plt.xlabel('time (s)')
9:  plt.ylabel('volts (V)')
10: plt.show()
```

输出图像，如图 6.17 所示：

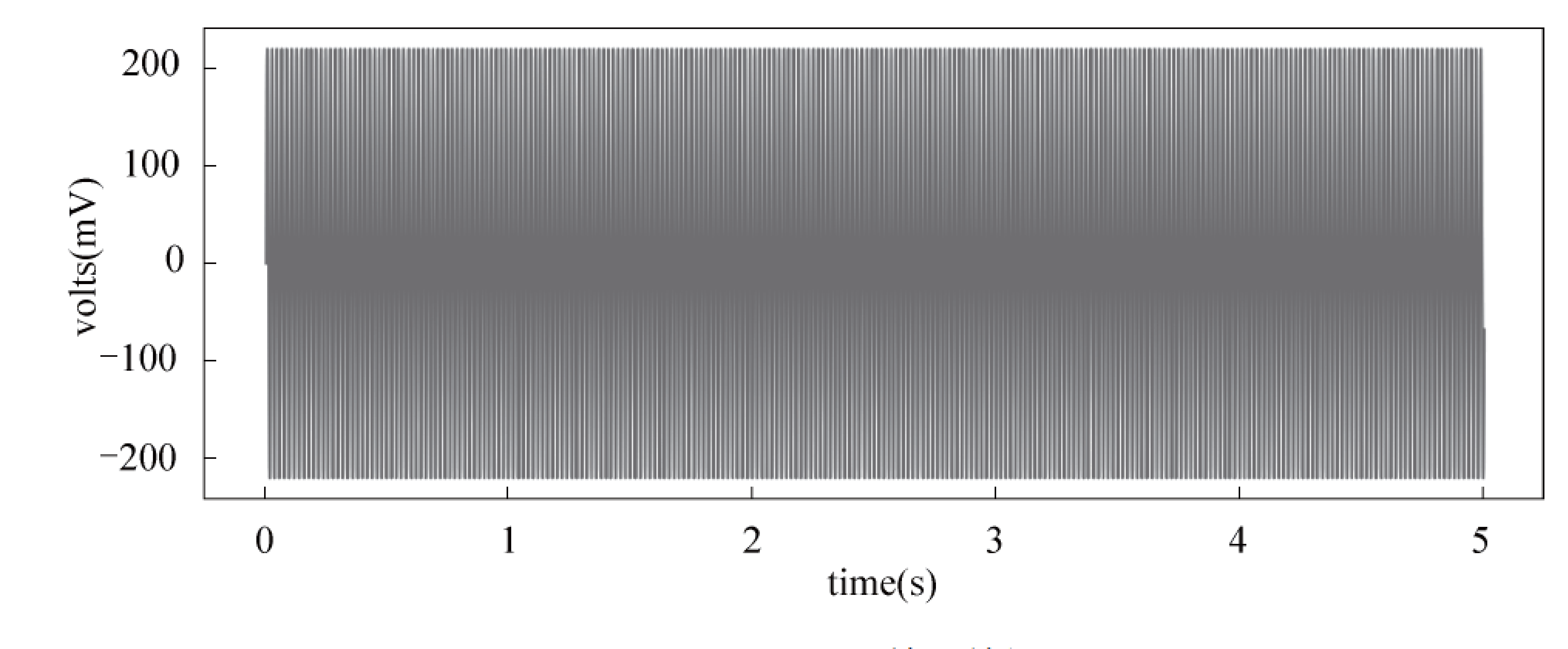


图 6.17　程序 6.10 输出结果

分析：

程序 6.10 中输出图像是前 5s 的电流波形。程序最重要的是第 4、5 行，下面先对这两行进行分析。np.arange(0.0, 5.0, 0.001)是使用 numpy 模块产生一个数据列表，这行代码的工作机制我们用下面这段代码解释。

```
import numpy as np
print(np.arange(1, 10, 3))
```

这段代码会输出[1 4 7]，这就很清晰地解释了 np.arange 的工作机制，它表示从 1 开始，每项加 3，直到累加的值超过 10 退出，它的范围是大于等于 1，小于 10。学习完这段代码

之后再看程序 6.10 的第 4 行，它创建了从 0 开始每 0.001 就产生一个值的一个列表，这段代码产生的列表内容为[0.000，0.001，0.002，0.003⋯0.999]。完成这段代码之后，用于画图的 x 轴数据已经完成并存放在变量 x 中。

接下来开始创建 y 轴数据，第 5 行代码 y=220 * np.sin(100*np.pi*x)表示为每个 x 中数据根据公式 $y = 220\sin(100 \cdot x)$ 生成 y 值，最后完成 y 轴数据的建立。完成这些后，使用 plt.plot(x, y)建立折线图。

看到输出图像图 6.17，你可能对这些折线图有些疑惑，我们使用了大量符合正弦规则的点来达到曲线的效果。可以使用交互界面的放大功能查看图像的细节，如图 6.18 所示是前 1s 内的波形，可以看出符合周期 0.02s，波形一共翻转了 100 次的规律。

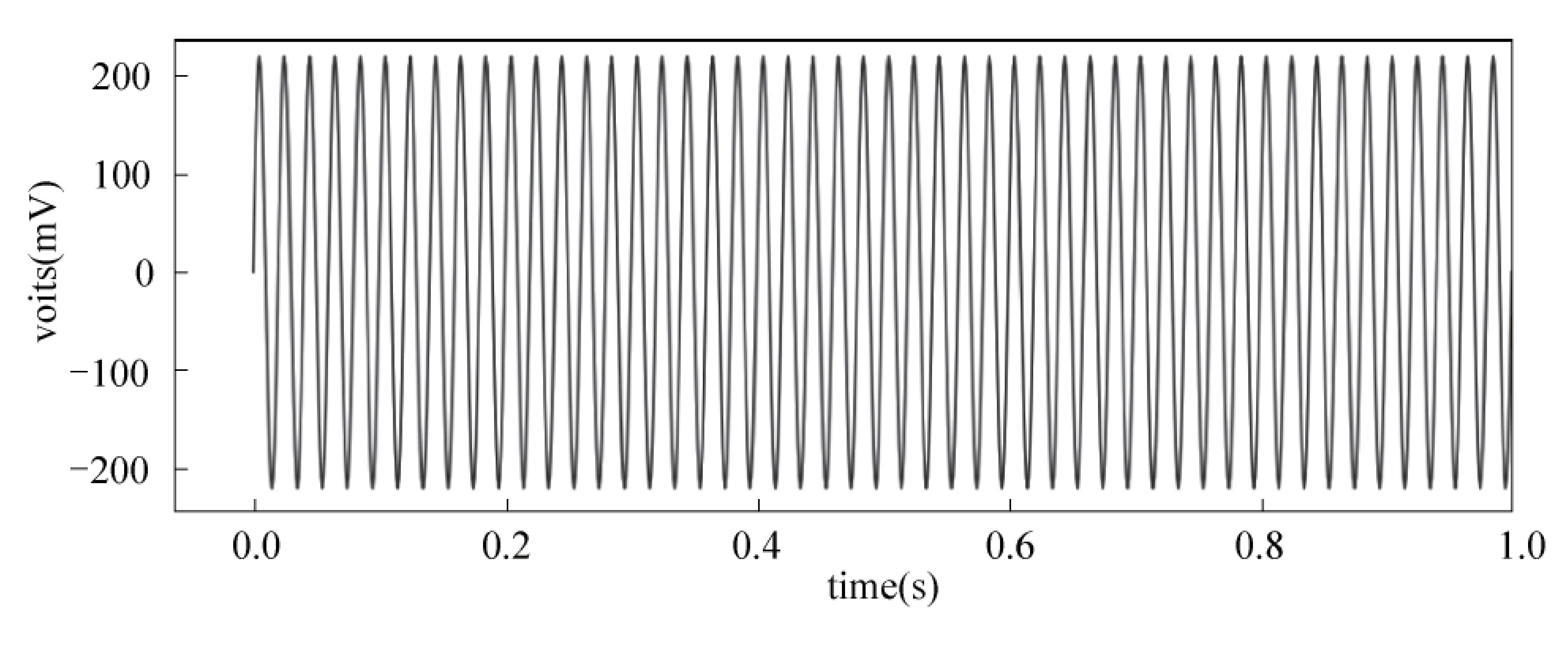


图 6.18　前 1s 内的波形

6.5　案例：统计文件字符出现频率

在这一节中我们来处理一个较为综合的案例，在第 5 章文件操作的 5.6 节讲到了将文件中字符出现的次数写到文件中，本节我们要利用这个文件中的数据，统计文件中出现频率最高的前 10 个字符并作图显示其概率。

在第 5 章的 5.6 节最后写入文件的时候，使用的是“字符加空格再加字符出现的次数”这种格式，这并不是我们临时决定使用的，而是经过慎重考虑之后使用的，因为在这里很容易犯错。这种格式是一种 CSV 文件格式的衍生形式，这样会让我们以很方便的形式读出。

那么什么是 CSV 文件？CSV 文件是指文件内容使用字符分隔值的形式存入，文件是用纯文本形式存储的。我们在存入时只是使用了 CSV 格式，并没用 CSV 的文件，因为程序产生的是 result.txt 文件，它只是在内容中使用 CSV 文件的格式，若是想要存成 CSV 文件，则修改程序 5.4 中的第 10 行文件名为 result.csv 即可。但是这并不重要，因为我们这部分对文件的操作，文件内容使用 CSV 格式就足够了。具体程序如程序 6.11 所示。

程序 6.11　统计文件字符频率：

```
import csv
import matplotlib.pyplot as plt

word = []
num = []
sum = 0

with open('C:/Users/dell.dell-PC/Desktop/result.txt','r') as file:
    items = csv.reader(file, delimiter=' ')
    for item in items:
        word.append(item[0])
        re_num = (int(item[1]))
        num.append(int(re_num))
        sum = sum + int(re_num)

x = []
y = []

for i in range(10):
    x.append(word[i])
    y.append(num[i] / sum * 100)

plt.bar(x, y)
plt.xlabel('word')
plt.ylabel('probability (%)')
plt.show()
```

输出图像，如图 6.19 所示：

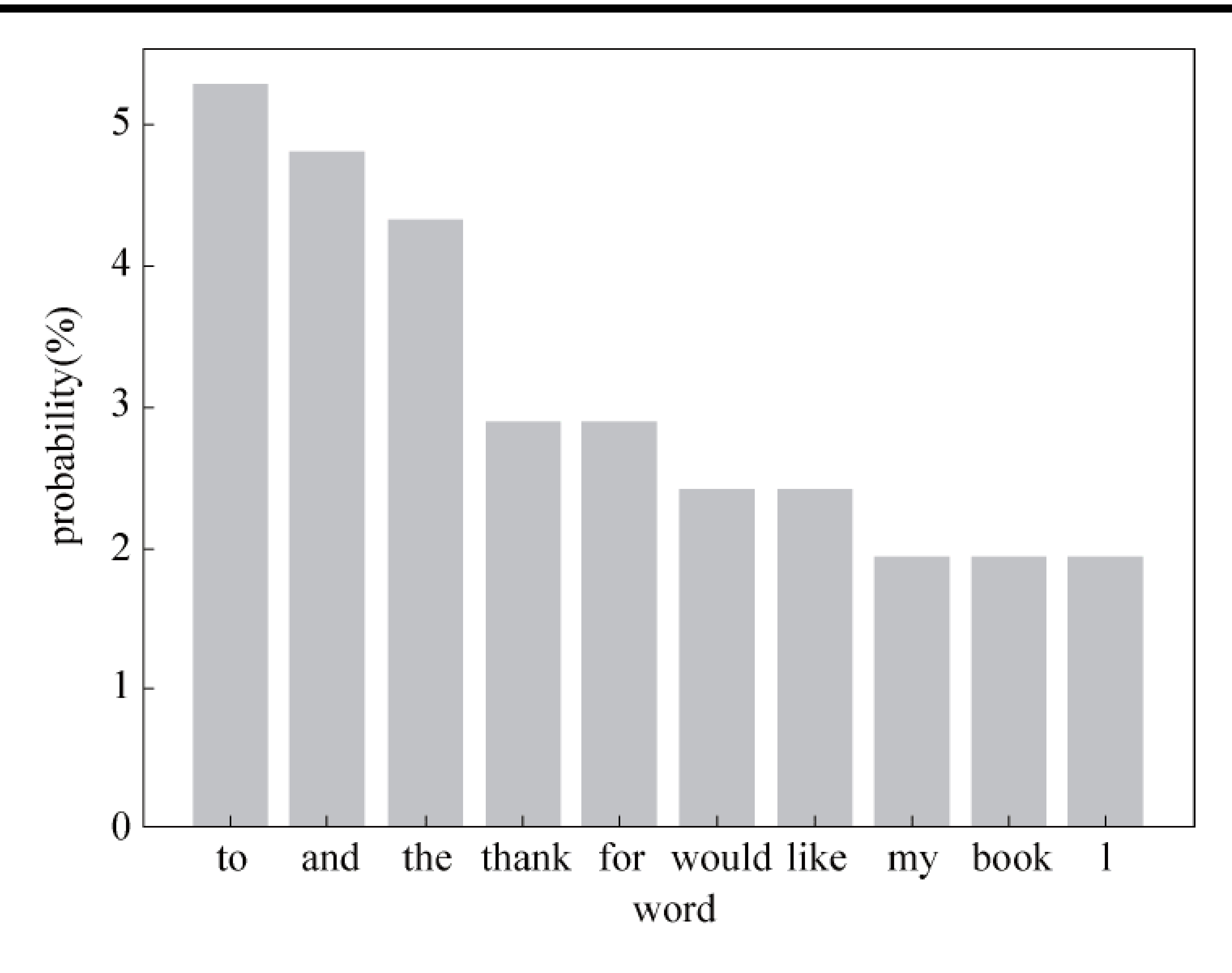


图 6.19　程序 6.11 输出结果

分析：

程序的第 1 行先将 csv 模块引入到程序中，程序的第 8 行使用的是 with open() as file 语句，完成了文件的打开并且文件的关闭会自动完成，同时，使用 file 获取返回的文件对象。使用 with 语句的好处在于如果执行过程中文件操作出现差错（如没有找到文件），它会报出一个与差错对应的异常。

完成文件的打开之后，我们将整个文件的内容使用 csv.reader(file, delimiter=' ')来读入，这条语句的意思是使用 CSV 格式来读取文件 file 中的内容，每行数据使用空格分隔符来区分。完成这些操作后，我们使用循环来遍历 items 也就是文件中的内容，而此时的 item 是文件中已经完成分割的具体每行的内容，例如，在第一次循环中，item 的值为<class 'list'>: ['to', '11']。在读入文件内容的过程中，Python 会自动将文件内容处理为字符串，因此在将数字存入 num 列表之前要先转为 int 类型。程序的第 14 行计算所有字符出现的次数并用于概率的计算。

至此，我们已经得到文件的所有内容，由于在存入文件时已经对出现次数排序，只需再次使用 for 循环取出前 10 条信息用于显示。将 *x* 轴数据设置为字符，*y* 轴为其出现的概率（出现次数/sum）。最后使用这些数据创建条形图并显示。

到这里程序已经完成了，这里再谈一个细节，第 5 章中的 5.6 节程序 5.4 的第 4 行是使用空格将整个文件分割，而程序 5.4 的第 12 行写入时也是使用空格分隔数据，程序 6.11 读

入时也是使用空格，这并不是巧合，当然了，写入文件和读入文件时的文件格式一定要相同，但是我们之所以都选择空格是因为程序 5.4 的第 4 行是使用空格将整个文件分割，这使得这个分割后的数据中已经没有空格，后续使用空格时不会造成任何冲突。

6.6　趣 味 练 习

在本章的趣味练习中我们使用 matplotlib 模块来读取并显示一张图片，这是简单但重要的第一步，因为它是计算机图像处理的第一步。不同于之前的是，我们还要添加 matplotlib 模块中的 image 来对图片进行处理。详见程序 6.12。

程序 6.12：

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

pic = mpimg.imread('C:/Users/dell.dell-PC/Desktop/pic.png')

plt.imshow(pic)
plt.axis('off')
plt.show()
```

输出图像，如图 6.20 所示：

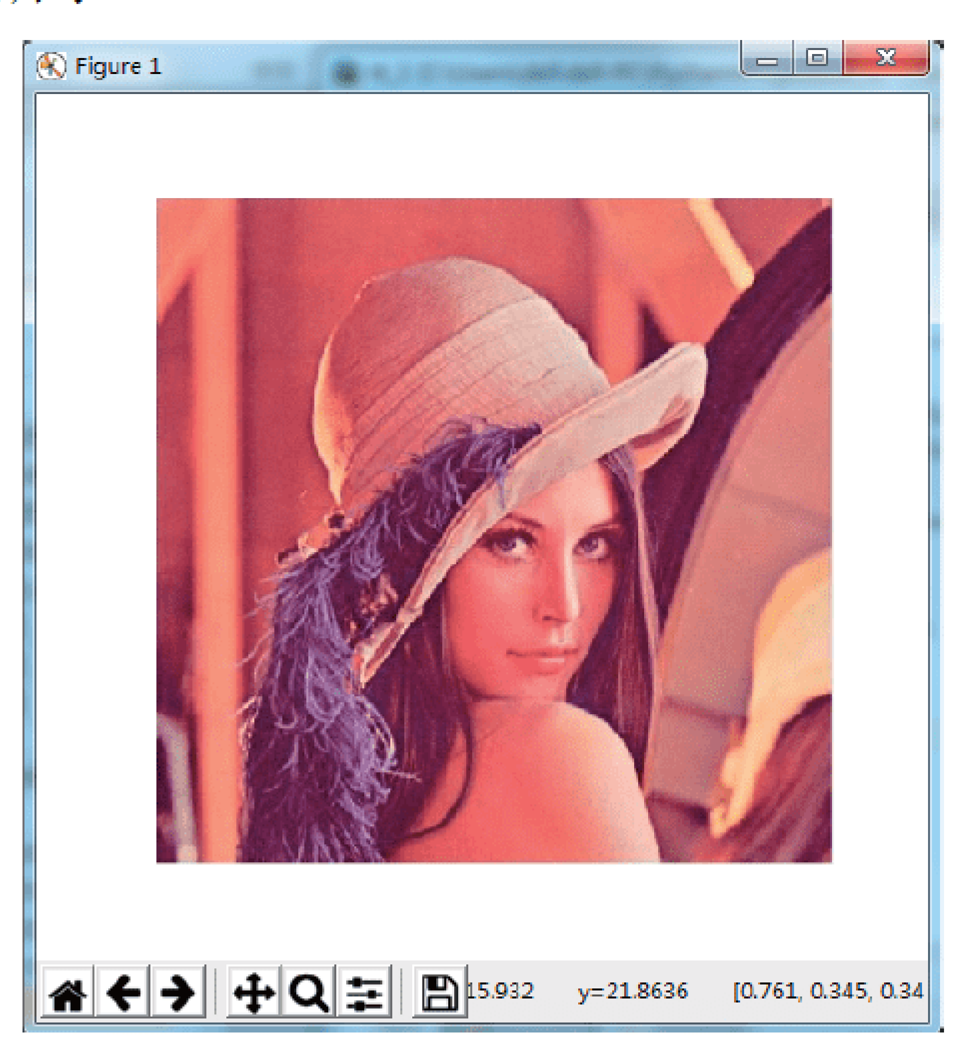


图 6.20　程序 6.12 输出结果

点睛：

程序的第 2 行将 image 引入到程序中，第 4 行使用 mpimg.imread()读取文件内容，得到文件内容之后，使用 plt.imshow()创建图片。因为图片显示没有用到坐标信息，因此使用 plt.axis('off')将显示窗口中的坐标关闭显示。最后 plt.show 在窗口中显示图片。

6.7　总　　结

本章学习了 Python 画图的知识，由于图形是一种十分直观的形式，当我们要处理信息比较或是观察信息变换等情况时，使用图像的形式展示出来会让我们对信息有一个直观的理解。因此，熟练地绘制不同形式的图形是很有必要的，它是我们的一个重要的工具。

6.8　练　　习

（1）简述 pandas 模块和 matplotlib 模块绘图的流程。

（2）使用你最近一次考试的各科成绩绘制饼图。

（3）使用列表[69, 55, 59, 40, 90, 87, 60, 86, 85, 71, 79, 91, 75, 76, 61, 63]和刻度列表[0, 60, 75, 90, 100]绘制直方图。

（4）使用 pandas 模块产生 1000 个 4 列正态分布的随机数作为坐标（x,y）中 y 轴的数据，x 轴数据为[0, 1, 2…999]，绘制 4 条折线图。

第 7 章　函　　数

到目前为止，书中涉及程序的功能大多都是由处于同一层面语句实现的。对于小型程序或是某些脚本，这是可以的。但是如果程序复杂，我们的工作量会变得庞大，同时，工作也会变得很无聊。

Python 为此提供了一种划分和组织程序的执行逻辑，通过这种机制，应用程序内容被划分成逻辑块的组合。这种机制被称为函数。通过本章的学习，需要掌握：

- ❑ 理解函数概念。
- ❑ 在编写程序时可以熟练使用函数。
- ❑ 明白函数参数传递的细节。
- ❑ 理解递归调用过程。
- ❑ 理解模块的工作过程。

7.1　什么是函数

我们对于函数并不陌生，在很早之前就在数学课中学过这一概念，先来回顾一下函数在数学中是如何描述的。“凡此变数中函彼变数者，则此为彼之函数”，也就是说函数指的是一个量随着另一个量的变化而变化，其中核心是两个量的对应法则。

在 Python 中，函数是将一些语句集合在一起，可重复使用的程序片段。它可以用来改变程序的某些状态，也可以用来处理一些数据或是用于计算出一个返回值。我们可以选择给函数传入或是不传入参数，并且参数的个数都是可选的。具体来说，其实函数只是将原来的编程模式按功能分块，并将所分的程序块改为可以广泛复用的工具，这样再遇到类似的功能时，只需要调用而不需要再次编写成段的代码。

为了下文的内容顺利进行，在这里我们先简单介绍一下函数的创建形式，关于它在编程中的具体形式及其相关的语法，在本章中会详细介绍。Python 中使用 def 加函数名来创建一个函数，如果函数有返回值的话可以使用 return 返回。先看看下面这段程序：

```
def y_line(x):
    y = 5 * x + 3
    return y
```

程序中使用 def 定义了一个名为 y_line 带有一个参数的函数，函数名后的 x 是要传入函数的参数，return 是将函数的计算结果 y 返回。

仔细想想，其实 Python 中的函数和数学课中学的函数是有一定联系的。在数学中，我们将变量的对应法则视为函数的核心，而对应法则实际上是对自变量如何影响因变量的一种描述。回看上述代码，Python 中函数的核心是对参数的处理并输出返回值（Python 函数中返回值是可选的）的过程，这对应数学中的自变量（参数）通过对应法则（函数中代码）得到因变量（返回值）。

现阶段对于 Python 中的函数可以先这样理解，但是，一定不要局限于上述数学函数的例子中，这只是帮助我们理解 Python 函数概念而已。在实际编程中，函数要比上文描述的强大得多。

7.2　为什么要使用函数

有关于这个概念我们在上节简单提到过，在这里再详细介绍一下。其实，函数有时也被称为子过程或子程序。那么为什么使用函数呢？其实函数在程序设计中主要扮演了如下两个角色：

（1）减少代码冗余和代码重用。简单地说就是一种打包逻辑，这样可以在之后的其他地方不止一次地调用。因为函数的这种一次编写多次运行的特点，我们的程序可以减少代码的冗余，这样也可以通过只修改函数中代码而达到将所有函数调用处都修改的效果。这样会使代码维护更为简单。

（2）流程分解。函数的机制会自然而然地将一个完整的程序流程分割成独立的子程序。实现较小的任务要比一次性完成整个流程要容易得多。函数讲的正是流程，说的是处理的过程。

函数不仅是减少冗余代码重用，流程分解，同时，它带来了新的编程模式，也给我们带来了很多关于编程思想的启示。

7.3　函数的创建和调用

7.1 节中简单地介绍了函数的创建，现在我们列出函数的一般形式：

```
def name(arg1, arg2,…, argN):
    <statements>
```

同之前讲的内容一样，函数也是使用 4 个空格的缩进来表示该函数的内容区域，也就是函数一般形式中<*statements*>所处的地方。def 是 Python 中函数创建语句，name 是要建立的函数的名称，用于之后调用时指明是哪个函数。首行不仅定义了函数名，同时，在函数名后使用括号包含了 0～N 个参数（因为参数没有具体值，称为形式参数，简称形参）。等到函数被调用时，将具体的值传入形参，用于函数代码进行相应的处理。

函数通常包含一条 return 语句返回函数处理的结果，如果函数中没有返回具体值的语句，Python 会自动添加 return None 语句，这个过程对我们是不可见的。下面列出带返回语句的形式：

```
def name(arg1, arg2,…, argN):
    ...
    return value
```

return 语句可以在函数主体中的任何地方出现，但是，return 表示本次函数调用的结束，程序也会返回到函数调用处继续执行。而 return 语句后的 value 给出函数的结果。下面通过程序 7.1 来看看函数的创建和调用。

程序 7.1：

```
def line_formula(a, b, x):
    y = a * x + b
    return y

formula_value = line_formula(3, 7, 2)
print("Equation's value is : {0}".format(formula_value))
```

输出：

```
Equation's value is : 13
```

分析：

程序中先用 def 创建了一个传入直线参数和 x 返回直线 y 值的函数，第 5 行中使用函数名调用函数。注意这个调用形式：函数调用时，不需要使用 def 了，直接用函数名加上要传给函数形参的值（由于这个值是一个明确的数，又被称为实际参数，简称实参），由于函数会返回 y 值，我们使用一个赋值语句将函数返回值传入 formula_value 中（使用 formula_value 接收函数的返回值）。这个函数比较简单，主要是用于熟悉函数的创建和调用。

注意：

在调用函数之前一定要先创建函数，否则 Python 会报出一个 name 'line_formula' is not defined 的错误。

7.4　作　用　域

在上一节中我们提到过函数是将一些语句集合在一起，可重复使用的程序片段，对于在这个片段中定义的变量必不可少的是要讨论其归属。在本节中我们要讨论的是 Python 中变量定义以及查找的区域，即变量的作用域问题。

在 Python 中一个变量的作用域（可以使用变量的地方）是由赋值的位置决定的，这是因为在赋值后 Python 的变量会在此地创建，而对于函数内的语句是一种可重复使用的程序片段，在这里定义的变量同函数之外定义的变量会有什么联系？在 Python 中，变量可以归纳为本地变量、全局变量和内置变量。先来看下面这一段程序：

```
x = 20
def p_str():
    y = x/10
    print(y)
p_str()
```

这段程序会输出“2.0”。从函数 p_str 的角度来说，x 是在它的上一级定义的变量，而在函数中通过赋值语句"="创建的 y 是函数中的本地变量。再进一步分析，在默认情况下，通过变量名赋值会创建或改变本地变量。在上述代码中，函数域中的变量 y 创建时使用了上一级的变量 x，这在 Python 中是允许的。同时函数中也可以定义同上级同名的变量，先看下面这段程序：

```
x = 10
def p_str():
    x = 99
    print(x)
p_str()
print(x)
```

这段程序会输出两行数，第 1 行是 99，第 2 行是 10。虽然函数中通过等号赋值语句创建的变量 x 和上一级定义的 x 是同名变量，但在 p_str 函数的作用域中声明的变量 x，只在函数内部起作用，在函数执行完毕后，函数内部的变量值对外部是无效的，全局变量并不会受到影响，所以外部的输出语句输出的依然是全局变量 x 的值。

上述这种使用未识别变量名的方式，Python 使用的是名为 LEGB 的机制。该机制其实很简单，所谓的 LEGB 是指：当遇到未认证变量名时，Python 依次搜索本地作用域 L，上层结构中的本地作用域 E，全局作用域 G，内置作用域 B，这个过程如图 7.1 所示。

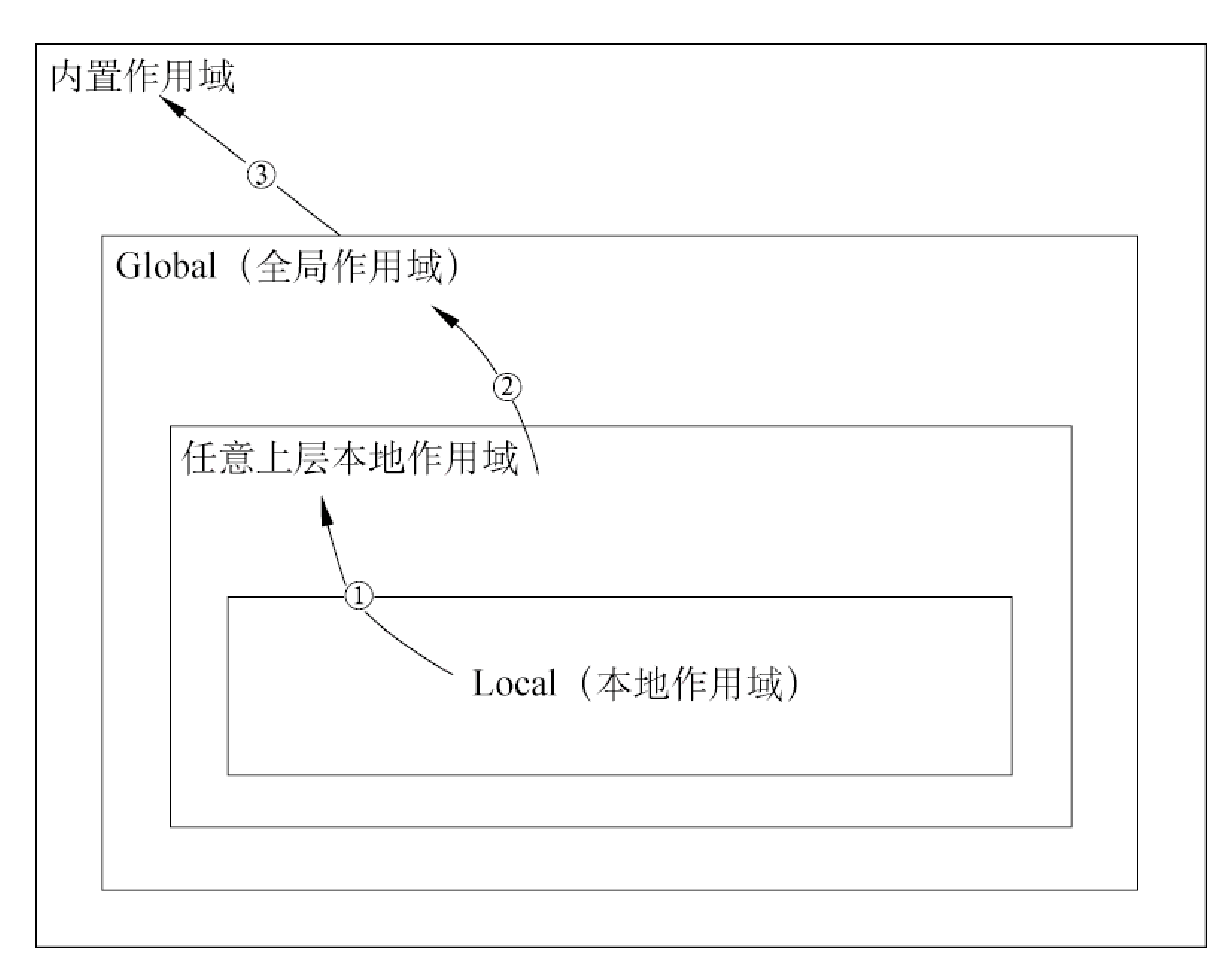


图 7.1　作用域查找原则（图中使用序号标识搜索顺序）

7.5　global 语句

global 语句的具体使用方式是通过关键字 global 后跟变量名，通过 global 语法声明的变量是位于文件内部顶层的变量。先来看看下面这段代码：

```
x = 10
def p_str():
    global x
    x = 99
    print(x)
p_str()
print(x)
```

这段代码会输出两行相同的内容，都是 99，输出的是函数内部变量的值。因为 p_str 函数中使用 global 语句使函数内的变量明确映射到全局，使用 global 关键字就是告诉 python 编译器这个变量不是局部变量而是全局变量，所以变量 x 在函数内被修改之后的值就映射到全局范围里，覆盖了最初定义的 x=10 的变量，即最后输出的都是被修改后变量的值 99。

注意：

尽管有些时候使用 global 语句对当前的情况是很方便的，但是，全局变量会使程序更难理解和使用。若是较多次地在函数中使用 global 语句，变量的变换由函数调用次数决定，而函数调用的顺序是由某种逻辑顺序或是其他顺序排列的，这样会导致调试程序变得非常困难。因此，全局变量使得程序逻辑变得复杂。在现阶段不熟悉编程的情况下，我们可以使用函数值传递的方式来处理改变值的问题。总之要尽可能避免使用全局变量。

7.6　参　　数

参数是函数用于接收外界信息的一种形式，这个参数由程序设计人员提供，函数接收到参数之后便可以利用这些值来做一些事情，之前我们已经提到过形参和实参的概念，在本节中，我们详细看看参数传递过程中的细节。

在编写函数的时候，我们将函数中的语句作用在形参上，待到函数运行时进行赋值，

将实参传入形参。这个过程看似简单，但其背后的细节我们要在这详细介绍一下。

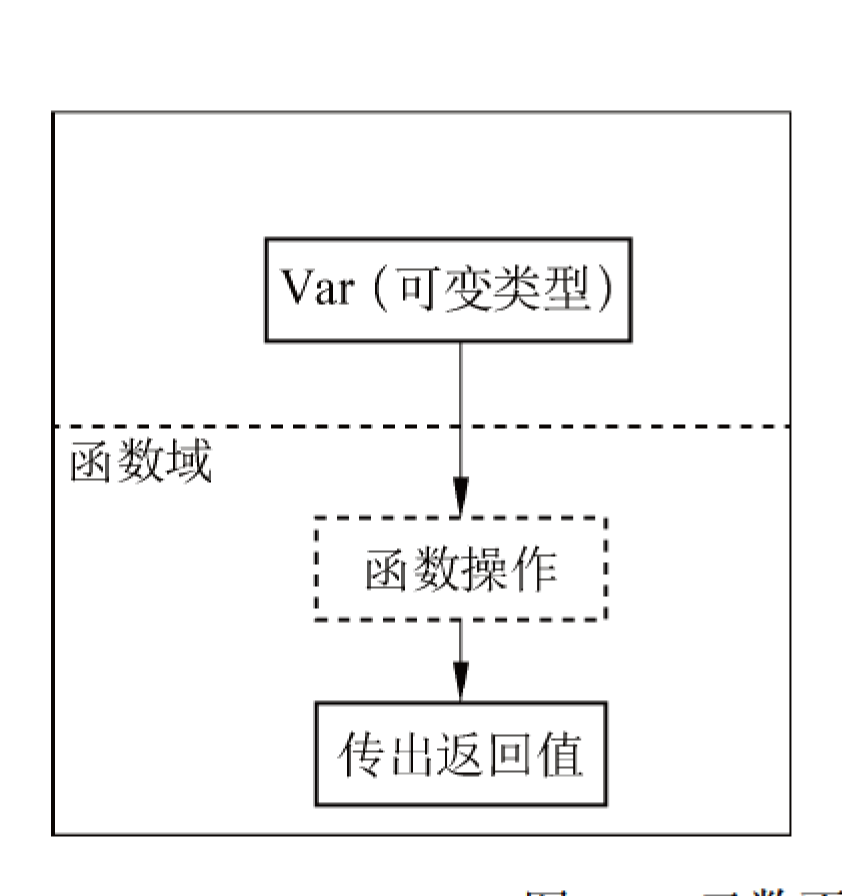
7.6.1　参数类型

之前在讲数据类型的时候提到过不可变类型和可变类型，由于在函数中我们要在传入的参数之上进行操作，所以在详细介绍有关函数的内容之前需要先明白参数类型的意义。

表 7.1　可变类型和不可变类型对照

可变类型	不可变类型
可变类型指的是变量在内存中存储的内容是可以改变的。由于这个特点，如果要对该变量本身进行操作只需要在原有的内存上进行操作。 可变类型包括字典型（dictionary）和列表型（list）	不可变类型指的是变量在内存中的内容是不允许修改的。因此当要改变这种类型的变量时只能为其创建新的对象，之后再将原值抛弃。 不可变类型包括数字、字符串和元组

了解可变类型和不可变类型之后再来看看函数参数传递时的细节，当函数参数传递的是可变参数时，可变类型本身传入函数中，函数中代码段直接操作在该变量上。另外，对于不可变类型参数的函数传递，当函数调用时，在函数中创建一个参数的复制对象，函数中代码段操作在该复制对象上，函数内操作并不会影响函数外部传递的不可变参数值。对于这两种类型的传递过程如图 7.2 所示。



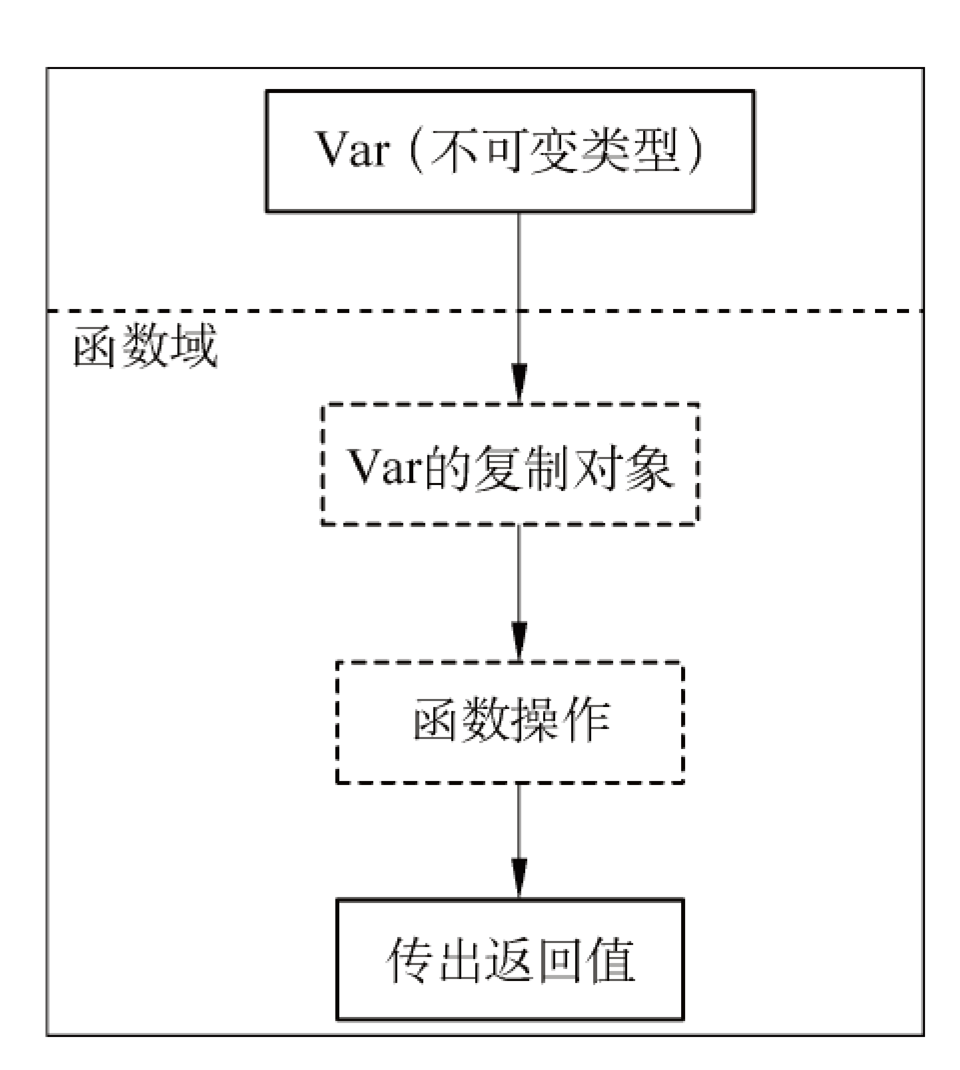


图 7.2　函数不同类型参数传递过程

7.6.2　浅拷贝和深拷贝

上一节中提到过函数参数传递的过程，其实这个过程有点像我们接下来要说的浅拷贝和深拷贝，在正式开始这节内容之前我们来讲一下常用的“=”赋值操作的内部。先看下面这段程序：

```
data1 = [1, 1]
data2 = data1
data2[1] = 2
print(data1, data2)
```

这段程序的输出是[1, 2] [1, 2]，没错，对变量 data2 的更新也会作用到 data1 上，这是因为 data2 =data1 这行代码只是给变量 data1 所指的内存起一个别名：data2，实际上 data1 和 data2 所指的为同一段代码。因为 data1 和 data2 代表的是同一段代码块，修改 data2 其实是修改 data2 指向的内存的内容，所以再用 data1 调用这段内存的内容时变化也会体现出来。代码中的 data2=data1 图解如图 7.3 所示。

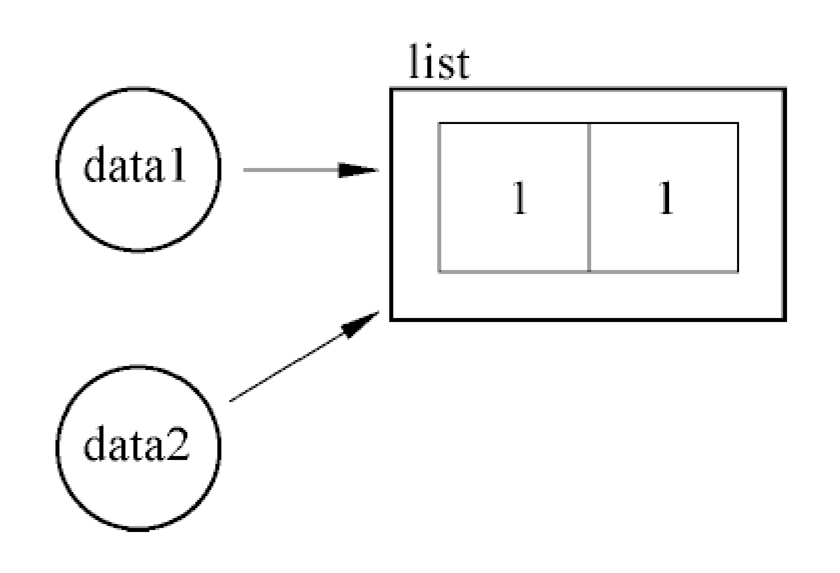


图 7.3　赋值操作图解

注意：

这里我们使用可变对象是因为不可变对象不能体现引用的特点，不可变对象使用“=”赋值之后内存中也是同图 7.3 这样，但是要改变其中一个变量的值时，由于不可变对象的特点两个变量不会都改变，而是会为改变动作发出的变量重新赋值。

1. 浅拷贝

浅拷贝只会拷贝内容的父对象，也就是顶层对象，不会拷贝对象内部的子对象。接下来看看下面的程序 7.2。

程序 7.2　浅拷贝示例程序：

```
1:    import copy
2:    data1 = [["data1", "This is data1"], 1]
3:    data2 = copy.copy(data1)
4:
5:    data2[0][0] = "data2"
6:    data2[1] = 2
7:
8:    print(data1)
9:    print(data2)
```

输出：

```
[['data2', 'This is data1'], 1]
[['data2', 'This is data1'], 2]
```

分析：

浅拷贝和深拷贝使用了 copy 模块，在编写程序之前要先将模块引入，关于模块的内容我们在后文再详细介绍。

程序的第 3 行使用 copy 方法将 data1 的内容浅拷贝到 data2，由于 data1 不是单层对象，其中的深层对象也就是列表["data1", "This is data1"]为 data1 和 data2 共享的内存内容，外层列表中的第二项数字 1 是双方独自拥有的。因此，改变 data2 子模块的值，通过结果输出可以看出对于深层对象 data1 和 data2 的内容都改变了，修改 data2 外层的对象数字仅有 data2 改变。关于浅拷贝 data2 = copy.copy(data1)这行代码的图解如图 7.4 所示。

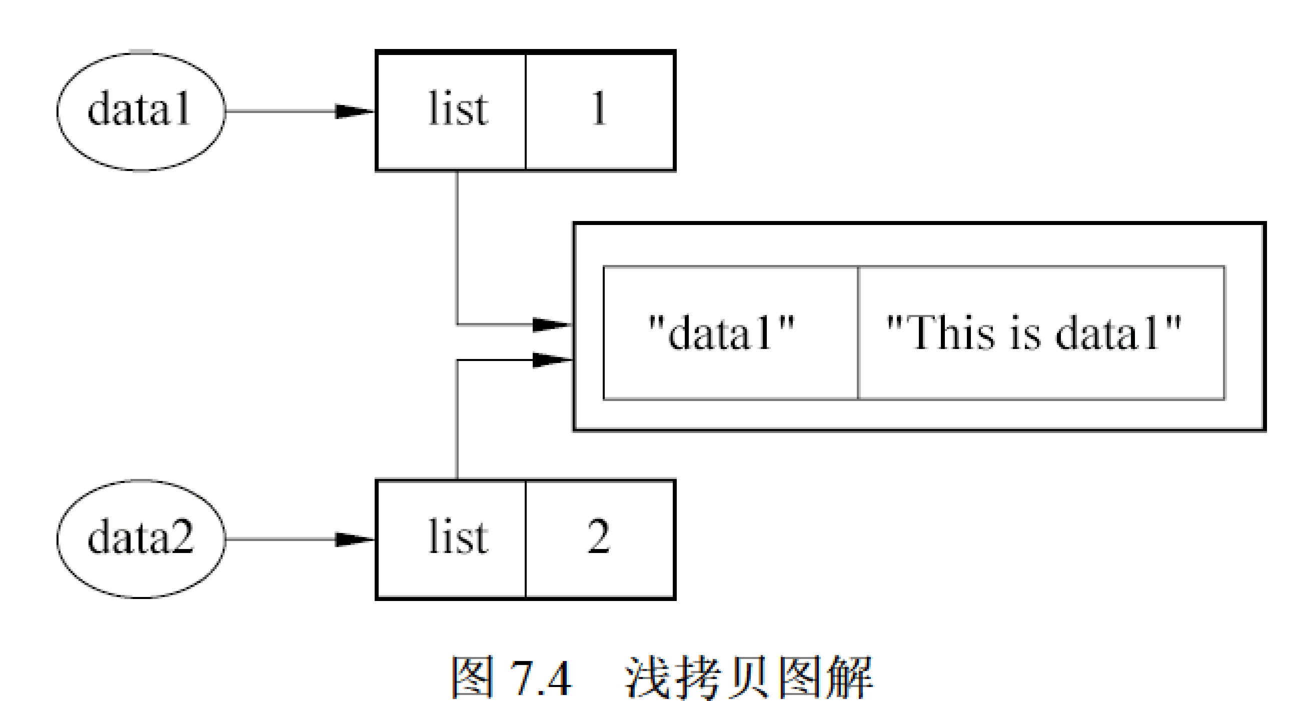


图 7.4　浅拷贝图解

2. 深拷贝

深拷贝使用 copy 模块中的 deepcopy 函数，它指的是拷贝时完全拷贝了父对象及其子

对象。完成拷贝后参与拷贝的变量是完全独立的。接下来看看下面的程序 7.3。

程序 7.3　深拷贝示例程序：

```
1:	import copy
2:	data1 = [["data1", "This is data1"], 1]
3:	data2 = copy.deepcopy(data1)
4:
5:	data2[0][0] = "data2"
6:	data2[1]=2
7:
8:	print(data1)
9:	print(data2)
```

输出：

```
[['data1', 'This is data1'], 1]
[['data2', 'This is data1'], 2]
```

分析：

程序的第 3 行，深拷贝将深层的子对象也从 data1 拷贝到 data2 中。data1 和 data2 就是两个拥有相同内容对象的独立变量，通过输出我们可以看出，修改 data2 中子对象的内容时 data1 不会受到影响。关于深拷贝的 data2=copy.deepcopy(data1)语句的图解如图 7.5 所示。

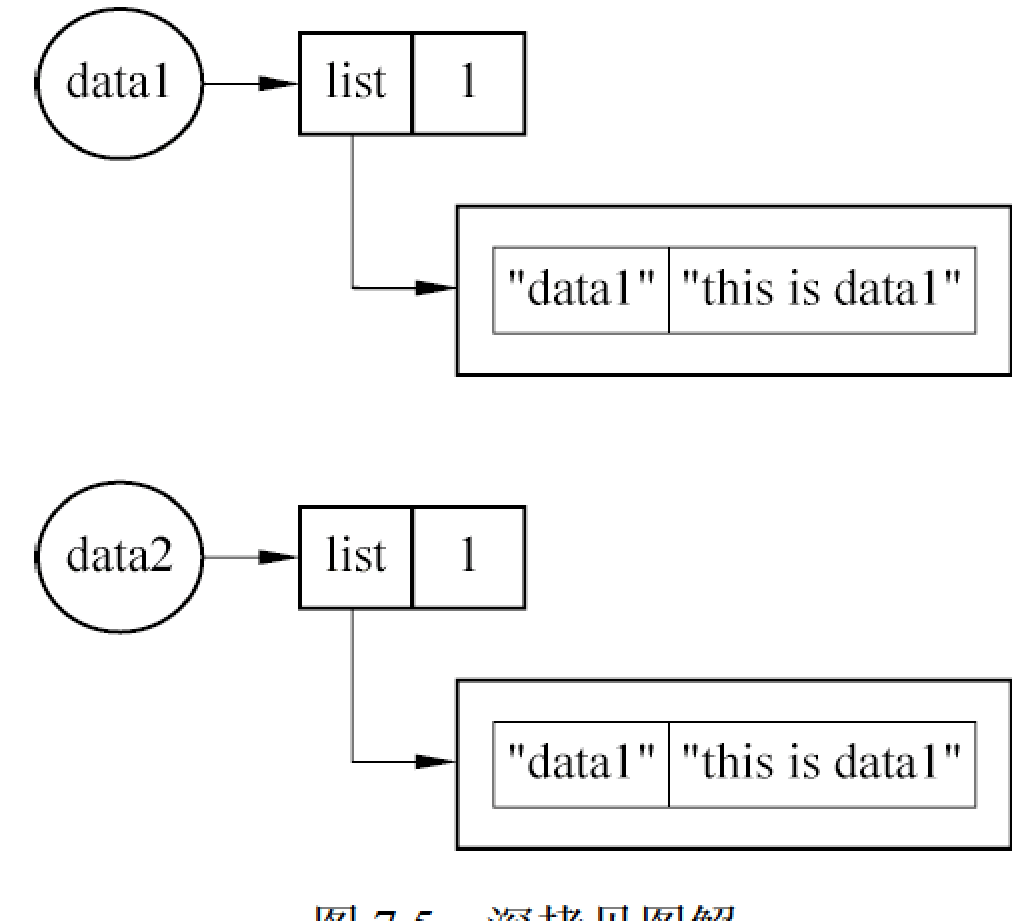


图 7.5　深拷贝图解

注意：

关于拷贝的知识先讲到这，在这里我们从可变对象和不可变对象的角度谈谈拷贝。对于不可变对象，其实没有拷贝的说法，因为当改变不可变对象时，都是用新对象替换旧对象。而可变对象的拷贝过程便是我们上文所谈的，分为浅拷贝和深拷贝。

7.6.3　函数参数形式

Python 中的函数参数形式很灵活，这使得在使用时会让我们有很多的自由空间。下面依次来介绍默认参数、缺省参数、关键字参数和可变参数。

1. 默认参数

默认参数是缺省参数的基础，当缺省参数时 Python 会自动分配默认参数值到函数，因此默认参数在程序运行时有很大的好处，它可以避免一些错误的发生。我们先通过下面这段程序来看看其使用形式。

```
def ptr(name = "None", age = 0, grades = 0):
print(name, age, grades)
ptr()
```

默认参数的最主要形式特点是函数在声明参数时使用“=”来为参数指定默认值。通过使用上面这种默认参数的函数声明后，使用 ptr()这种函数调用方法是完全没有问题的，这段程序会输出 None 0 0。

2. 缺省参数

有了默认参数的基础，缺省参数便很容易理解了，先来看下面的程序。

```
def ptr(name = "None", age = 0, grades = 0):
print(name, age, grades)
ptr("Leo", 1)
```

这段程序因为使用了默认参数，在函数调用时不必将参数全部指出，程序会输出 Leo 1 0。调用时使用了缺省参数，因此 grades 为默认值 0。这段程序没有什么难点，但是要注意的是这种调用方式是通过位置的方式将形参和实参匹配的。那是否有一种更为灵活的方式？让我们继续来看看关键字参数的方式。

3. 关键字参数

为了摆脱之前说到的位置匹配的缺点，Python 允许通过变量名进行匹配，但是它的形

式比之前略为复杂。先看下面这段程序。

```
def ptr(name = "None", age = 0, grades = 0):
print(name, age, grades)
ptr("Leo", grades=1)
```

这段程序通过使用关键字匹配的方式跳过 age，直接指定变量 grades 的值。变量 name 的赋值是通过位置指定的，同时也使用了缺省参数。它们混在一起也充分体现了 Python 函数的灵活性。

4. 可变参数

当我们的函数要处理任意数量参数时，之前提到过的参数类型都不能胜任，Python 为这种情况提供了一种可变参数的形式，先来看下面的程序 7.4。

程序 7.4：

```
1:   def ptr(Name="Leo", *nums, **score):
2:       print("Name : {0}".format(Name))
3:
4:       for item in nums:
5:           print("No.{0}".format(item))
6:
7:       for part1, part2 in score.items():
8:           print(part1, part2)
9:
10:  ptr("Leo", 1, 2, 3,math=95, physics=92, chemistry=91)
```

输出：

```
Name : Leo
No.1
No.2
No.3
math 95
physics 92
chemistry 91
```

分析：

当程序的第 10 行调用函数时，实参传递给对应的形参，此时实参被分解为 Name:'Leo' num:<class 'tuple'>: (1, 2, 3)　score: {'math': 95, 'physics': 92, 'chemistry': 91}。实参之所以被分

解是因为形参中的“*”。具体来说，*nums 用于接收变量并存在元组中，**score 用于接收变量并存到字典中，字典的键和键值不能通过“：”链接，要通过“=”链接。当它们同时使用时，Python 会自动通过元素的形式分到不同的组中，但是不同类型的变量不能相互交叉（即不能使用类似于 1,math=95 这种方式）。

注意：

当函数的参数同时有*nums 和**score 时，*nums 一定要放在**score 前面，否则会报错。

7.7　递　　归

递归算是一个较为高级的话题，但是递归在使用时不一定是最简单的或是最高效的，并且，递归结构并不容易理解，所以在实际编程中较为少见。

递归指的是在函数的定义中使用函数自身的方法，即在函数中调用自身。拥有这种特点的函数被称为递归函数。刚一开始就罗列一大段晦涩的知识点不是本书的特点，下面给大家唱一首儿歌：从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事！故事是什么呢？“从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事！故事是什么呢？”……

看完递归概念再看这首儿歌有没有一种恍然大悟的感觉，就是这么相似，这么巧！但是递归有一点不同的地方，它是自定义条件退出。接下来看一个简单的求和程序 7.5，体味一下递归概念。

程序 7.5：

```
def sum(nums):
    print(nums)
    if not nums:
        return 0
    else:
        return nums[0] + sum(nums[1:])

print(sum([4, 2, 1, 8]))
```

输出：

```
[4, 2, 1, 8]
[2, 1, 8]
[1, 8]
[8]
[]
15
```

分析：

程序的第 6 行通过调用本身实现了递归操作，同时，第 3 行通过 if 语句设置了递归的退出语句。由于递归的过程较为复杂，我们直接使用图解的方式对程序进行分析，如图 7.6 所示。

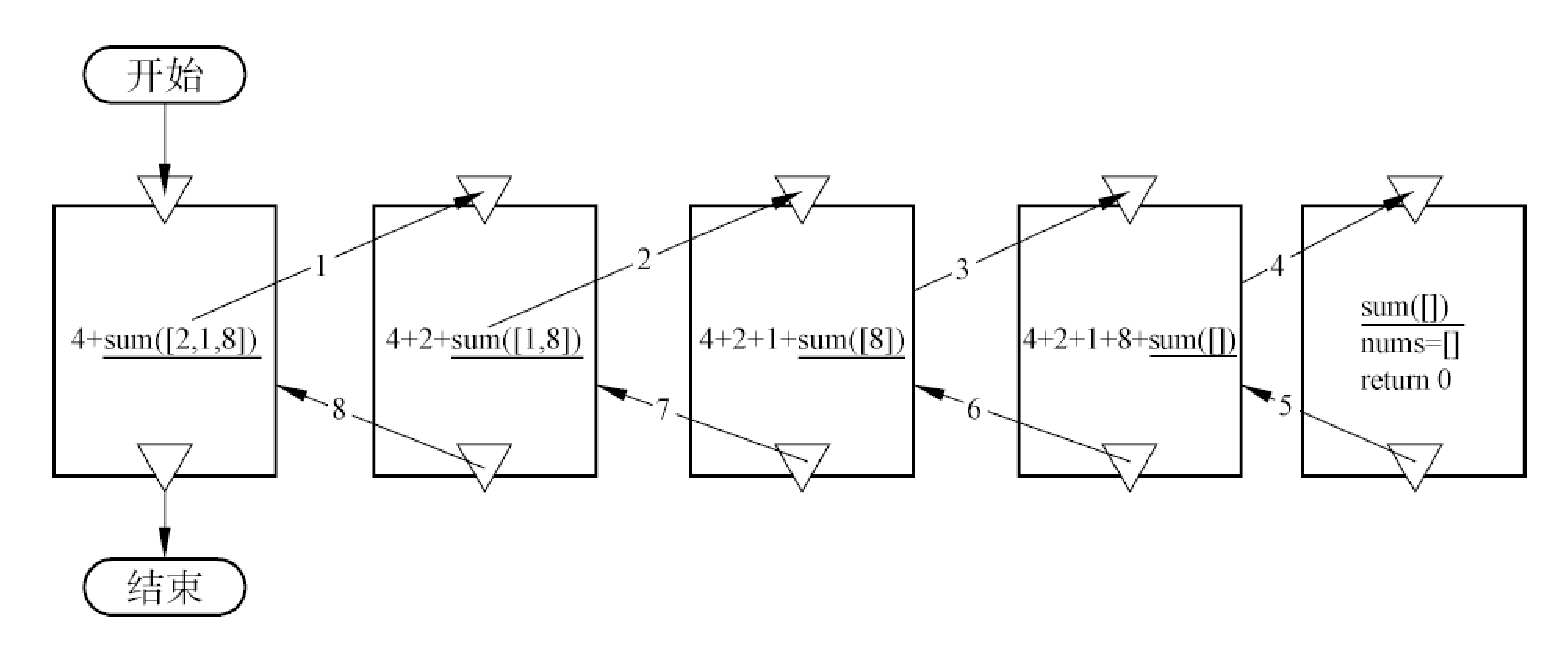


图 7.6　递归图解

每次函数运行程序的第 6 行都会用 sum(nums[1:])带着新的参数再次进入本函数中，直到新的参数为空，这时会执行程序的第 4 行的 return 0 使得这个递归依次返回并完成数值计算，这个过程在图 7.6 中用序号体现。

完成了上述递归求和的学习后，我们再来看看更加复杂的递归求 Fibonacci 数列各项的程序。Fibonacci 数列又称黄金分割数列，它指的是：0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,377…该数列从第 3 项开始，之后的每一项都等于前两项之和。该数列的递归程序实现如程序 7.6 所示。

程序 7.6：

```
def fibo(n):
    if n <= 1:
```

```
3:        return n
4:    else:
5:        return (fibo(n - 1) + fibo(n - 2))
6:
7: for i in range(10):
8:     print(fibo(i))
```

输出：

```
0
1
1
2
3
5
8
13
21
34
```

分析：

同上一个递归求和的函数相比，Fibonacci 数列比较复杂。但是实际上递归只是在求当前数列的项，而且每次都是要从第一项 0 算起，一直算到当前项，因此要使用 for 循环来控制输出数列的个数。正是因为每次都是从第一项 0 算起，该程序的效率较低。

程序的第 5 行使用了两次递归，但是这两次调用并不是平行顺序，它是一种类似深度优先的顺序。由于递归过程比较复杂，仅列出 fibo(4)的递归计算图解（见图 7.7）。

在图 7.7 中使用序号标识递归调用的顺序。虽然程序的第 5 行 return (fibo(n−1) + fibo(n−2))中两个 fibo 函数是并列的，但是实际是第一个递归完成后才进行第二个的递归。图中仅是数列中的第 4 项的计算过程，之所以图中的过程（a）中最下面 fibo(1) = 1 和 fibo(0) = 0 没有继续递归下去，这正是因为程序的第 2 行使用 if 设置了递归退出条件。

明白了程序递归运行机制之后我们会发现，相比较于循环计算 Fibonacci 数列的方法，即保留前两项值直接进行计算，递归计算过于烦琐，尤其是数列中靠后项的计算。因此，在选用递归做程序设计时要慎重。

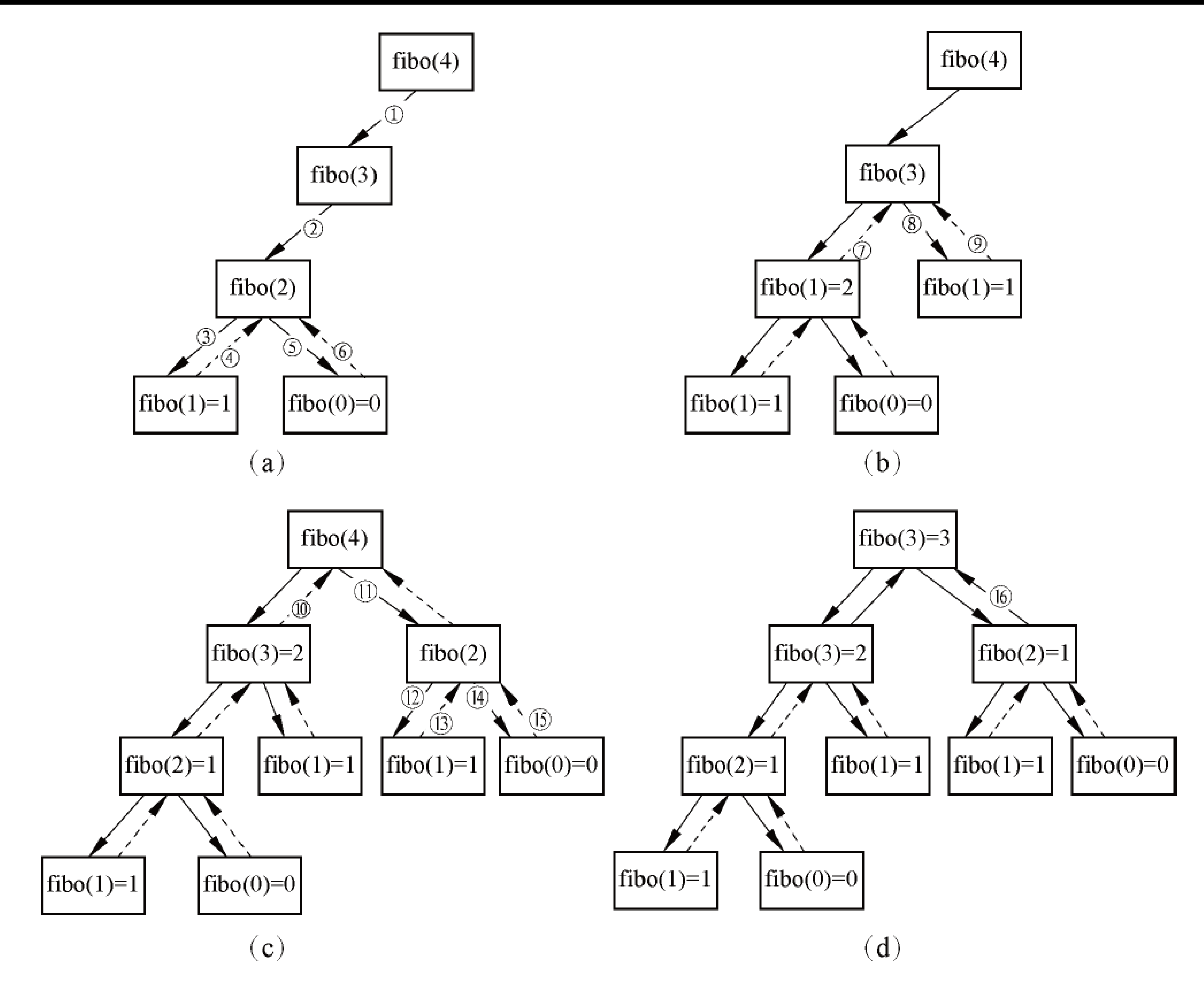


图 7.7　fibo(4)的递归图解

7.8　模　　块

至此，我们已经明白了如何在程序中通过定义函数来减少工作量，但是目前都是仅仅局限在一个 Python 文件中，而在实际应用时所用程序都在一个文件中编写是不太可能的，为了扩大函数重用的范围，我们引入模块这一概念。

7.8.1　什么是模块

我们对模块其实并不陌生，前面已经使用多次了，本章讲拷贝时也提到使用 import copy 语句引入 copy 模块。通过图 7.8 copy 模块的内容，可以看出模块中有很多我们熟悉的内容，

包括一些变量、函数的具体实现（有些还没有提到过的关键字会在后文中学习），其中就有我们之前使用的 copy 函数和 deepcopy 函数。

我们可以使用 dir 函数得到模块中内容列表，如语句 print(dir(copy))。这条语句会输出 copy 模块中定义的全部内容的名称：

```
['Error', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_copy_dispatch', '_copy_immutable', '_deepcopy_atomic', '_deepcopy_dict', '_deepcopy_dispatch', '_deepcopy_list', '_deepcopy_method', '_deepcopy_tuple', '_keep_alive', '_reconstruct', 'copy', 'deepcopy', 'dispatch_table', 'error']
```

```
class Error(Exception):
    pass

error = Error   # backward compatibility

try:
    from org.python.core import PyStringMap
except ImportError:
    PyStringMap = None

__all__ = ["Error", "copy", "deepcopy"]

def copy(x):
    '''Shallow copy operation on arbitrary Python objects.

    See the module's __doc__ string for more info.
    '''

    cls = type(x)

    copier = _copy_dispatch.get(cls)
    if copier:
        return copier(x)

    try:
        issc = issubclass(cls, type)
    except TypeError:  # cls is not a class
        issc = False
    if issc:
        # treat it as a regular class
        return _copy_immutable(x)

    copier = getattr(cls, "__copy__", None)
    if copier:
        return copier(x)
    reductor = dispatch_table.get(cls)
```

图 7.8　copy 模块内容（Error 部分的内容）

接下来详细分析下 import copy 是如何在程序中起作用的。当程序运行到 import copy 这一语句时，它便会开始寻找 copy 模块，这一语句就是告诉 Python 我们在程序中要使用该模块的内容。copy 是 Python 中的内置模块，我们不必指明模块目录，Python 知道它的位置。

另外，还有一条引用语句 from…import，它是将模块的部分内容引到我们的程序中。先来看下面这段程序：

```
from math import sqrt
print("√25：   ", sqrt(25))
```

程序将根号函数 sqrt 从 math 模块中引入程序中并使用它，这种引入方式不必使用模块名加点符（如 copy.deepcopy(data1)），可以直接使用函数名，但是这也带来了名称冲突问题。这样只引入模块的部分内容减少了计算机的工作量，但是，一般来说，我们在程序中应该尽量避免使用 from…import 语句，因为这条语句可能会导致程序中出现名称冲突并且让我们的程序更难被别人理解。

7.8.2　自定义模块

通过上述讲解大家对模块可能还是感觉有点神秘，如果你有保存之前的案例或是练习的程序，那么无形之中已经定义了很多模块。每一个独立的 Python 程序都可以当成一个模块。接下来我们使用代码段的方式正式创建一个模块并调用它。

首先使用之前学到的递归输出 Fibonacci 数列创建模块文件 mymodule.py，其内容如下：

```
#fibo 数列
def fibo(n):
    if n <= 1:
        return n
    else:
        return (fibo(n - 1) + fibo(n - 2))

def ptrFibo(n):
    for i in range(n):
        print(fibo(i))

__version__ = '0.2'
```

文件中的大部分内容都和之前的一样，更改部分只是创建了一个 ptrFibo 函数并将控制

输出数列个数的循环代码装入。最后添加了一行表示模块当前版本的变量，让该文件的内容更像一个模块。

接下来在模块文件 mymodule.py 同目录下创建调用模块文件 callModule.py，其内容如下：

```
import mymodule

print('The version of this module is : ', mymodule.__version__)
mymodule.ptrFibo(10)
```

调用程序中，我们使用模块名加点符使用模块中的变量和函数。那么 Python 是如何找到我们定义的模块的呢？在创建 callModule.py 文件时我们提到过要在同目录下创建，这是因为当导入模块时 Python 会按照“当前目录→PYTHONPATH（安装过程中的默认路径）下的每个目录→默认路径（系统中的默认路径）”这个顺序搜索模块直到找到该模块。

7.9　趣 味 练 习

本次的趣味练习将使用 Python 做一个简单的绘图板，我们使用 Python 监听键盘中的上下左右按键并对它做出相应的反应。先来看程序 7.7。

程序 7.7：

```
import turtle
t = turtle.Pen()
t.turtlesize(2, 2, 2)
def up():
    t.forward(50)
def left():
    t.left(90)
def right():
    t.right(90)
def down():
    t.right(180)

turtle.onkeypress(up, "Up")
turtle.onkeypress(left, "Left")
turtle.onkeypress(right, "Right")
```

```
16:   turtle.onkeypress(down, "Down")
17:
18:   turtle.listen()
19:   turtle.done()
```

输出图像，如图 7.9 所示：

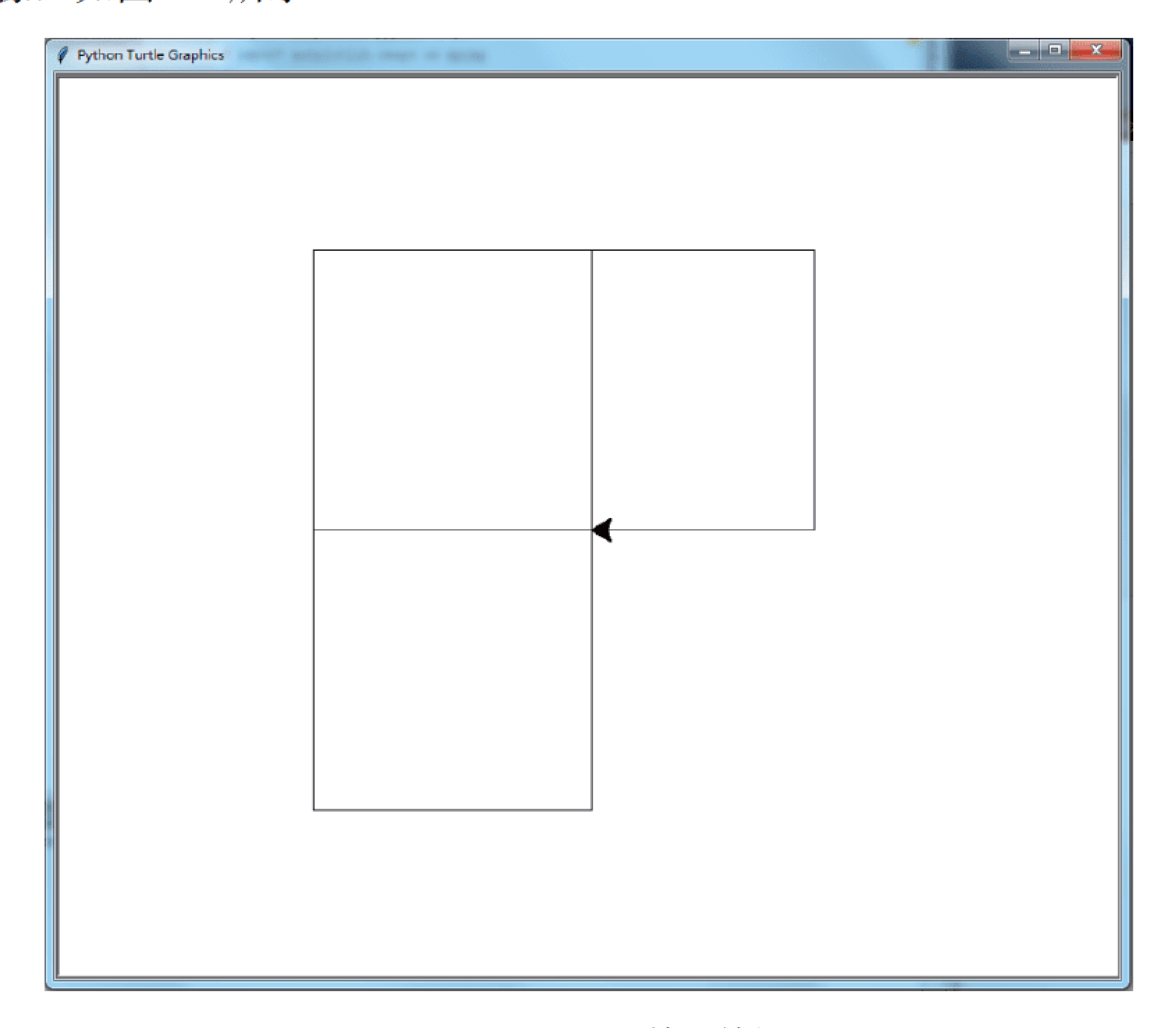


图 7.9　程序 7.7 输出结果

点睛：

对于 turtle 模块我们已经非常熟悉了，在这里只对程序中部分内容进行分析。程序的第 3 行 t.turtlesize 的作用是设置绘图时的方向指示箭头，其中前两个参数分别是箭头的长和宽，第 3 个参数是箭头的圆滑程度。

接下来对应键盘中的 4 个方向键设置 4 个函数用于箭头的转向，但是因为绘图中不能只转向，因此我们将前进箭头对应的函数设置为向前画出 50 个单位的直线。定义完函数之后，我们使用 onkeypress 将其绑定到按键上。最后使用 turtle.listen()监听键盘的输入，语句 turtle.done()用于关闭 turtle，如果没有这条语句的话程序可能会闪退。

7.10　总　　结

本章主要学习了关于函数的操作以及 Python 中函数机制的细节，其中最重要的是将函数这种思想应用到程序的编写中。本章还学习了有关模块的知识，这些对于编写中大型程序都是必不可少的，并且，不管是自己编写程序还是阅读引用已有的库，函数都是很重要的功能实现手段。本章还介绍了递归的概念，这部分内容虽然难度比较大，但是希望大家可以掌握这些概念。

7.11　练　　习

（1）编写一个无返回类型的函数，将圆的半径作为参数传入函数中，要求函数的功能是计算圆的周长和面积。

（2）观察下面这段程序，写出函数 space 中的可用变量。

```
total = 0
x = 10
def p_str():
    y = 99
    global n = 1
    n += 1
    print(x)

def space():
   print()

p_str()
print(x)
```

（3）观察下面这段程序，写出程序的输出结果，并画图分析为什么数字为不可变类型最后却被改变了。

```
import copy
data1 = [[58, "This is data1"], 1]
data2 = copy.deepcopy(data1)
```

```
data2[0][0] = 89

print(data1)
print(data2)
```

披荆斩棘（选做）：

向程序中输入一个数 n，并使用递归求出 n 的阶乘。要求画出递归调用简图。

第 8 章　面 向 对 象

在此之前的全部案例、练习都是以处理问题的过程为中心，我们所编写的程序大多是些处理数据的代码块。这其实是一种名为面向过程（Procedure-oriented）的编程方式。早在本书的开始就提到过 Python 支持面向对象编程，那么到底什么是面向对象？本章我们将学习具体的面向对象编程技术。先在这里强调一下，本章内容是 Python 的重点，而且相比前序章节较为抽象，难度较大，希望大家重视。

通过本章学习，需要掌握：

- ❑ 理解面向对象的相关概念。
- ❑ 类及其相关操作。
- ❑ 继承概念及其实现方法。
- ❑ 如何设计继承结构。

8.1　面向对象与面向过程

面向对象和面向过程是两种不同的组织程序的方式，在 Python 中面向对象是可选的，在学完本章后，我们不必强迫将所有程序都转为面向对象，但要尽快让自己适应这一模式。首先来看看面向过程和面向对象分别是什么，它们有什么区别。

8.1.1　面向过程

什么是面向过程？从字面意义上理解，面向过程就是用解决问题的具体过程进行编程。仔细思考一下，在学习和工作中，当我们去完成某个方程的求解时，一般会按照公式的先后顺序依次求解自变量。这种按照先后步骤去解决问题的方式，实质上就是按照面向过程的思想去解决问题。我们运用的公式就是过程，按照步骤解决问题就是面向过程。面向过程编程思想的一般实现步骤如下：

（1）将要实现的功能描述为一个有着先后顺序的连续步骤。

（2）依次完成这些步骤，如果某一步的难度较大，又可以将该步骤细化为若干个子步

骤，以此类推，一直到获得想要的结果。

（3）步骤的主体是函数，一个函数就是一个已封装的模块，可以实现一定功能，各个子步骤通过函数完成封装，从而实现代码的重用和模块化编程。

8.1.2　面向对象

接下来看看面向对象。在现实世界中，任何一个操作或是业务逻辑的实现都需要一个实体来完成，即实体是动作的支配者，没有实体就没有动作。所谓的面向对象，就是用模拟现实世界的思维去编程，按照现实世界中的逻辑去分析问题，思考问题中涉及哪些实体，且这些实体具备哪些属性和方法（属性可以看作对象的数据，方法可以看作对象的动作。关于属性和方法我们在下一节中细谈），我们又该如何设计并调用这些实体的属性和方法去解决具体的问题。

所谓的模拟现实世界，就是使计算机的编程语言在解决相关业务逻辑时的方式，与真实的业务逻辑的发生保持一致，即每一个动作的背后都有一个完成这个动作的实体，这样任何功能的实现都依赖于一个具体实体的动作，因此面向对象可以看作是一个又一个的实体在发挥其各自“能力”并在内部进行协调有序地调用过程。当采用面向对象的思想解决问题时，可分为下面几步：

（1）分析哪些动作是由哪些实体发出。

（2）定义这些实体，为其增加相应的属性和方法。

（3）实体去执行相应的功能或动作。

关于上述所说的面向对象内容我们再多说几句，就以流水线的机器人为例，位于不同阶段的机器人有着不同的作用，它们有着很多属性（如机器人的 id 或是标识信息），也有一些方法（如焊接机器人有焊接方法，喷漆机器人有喷漆方法），这些完成不同工作的机器人其实就是我们上文说的实体，而将它们抽象出来的便是类。类用于表示机器人有哪些属性和方法。我们可以想象为机器人制造商就是参照类来生产机器人。

8.2　类

类是 Python 面向对象的主要工具，它使用 class 语句建立。在本节中我们详细谈谈关

于类的相关知识。

8.2.1　类和实例

通过上节末尾的内容介绍，我们对类已经有了一定的认识。类是用来描述具有相同的属性和方法的实例的集合，它包含了这个类的所有实例所共有的属性和方法。

类和实例的关系是很密切的，但是一定要注意的是，它们是两种不同的类型。接下来我们来谈谈类和实例的差异。

我们也可以将类理解为一种产生实例的工厂，不同的工厂（类）可以创造出不同的产品（实例）。只要有需要，通过类制作多少实例都是允许的。例如，现有一个台式计算机的类，它有三个属性和一个方法。其中 CPU、内存容量和存储容量为属性，运行是方法。我们可以用这个类实例化出多台不同配置的台式计算机，图 8.1 描述的正是台式计算机类和这个类的一个实例化。

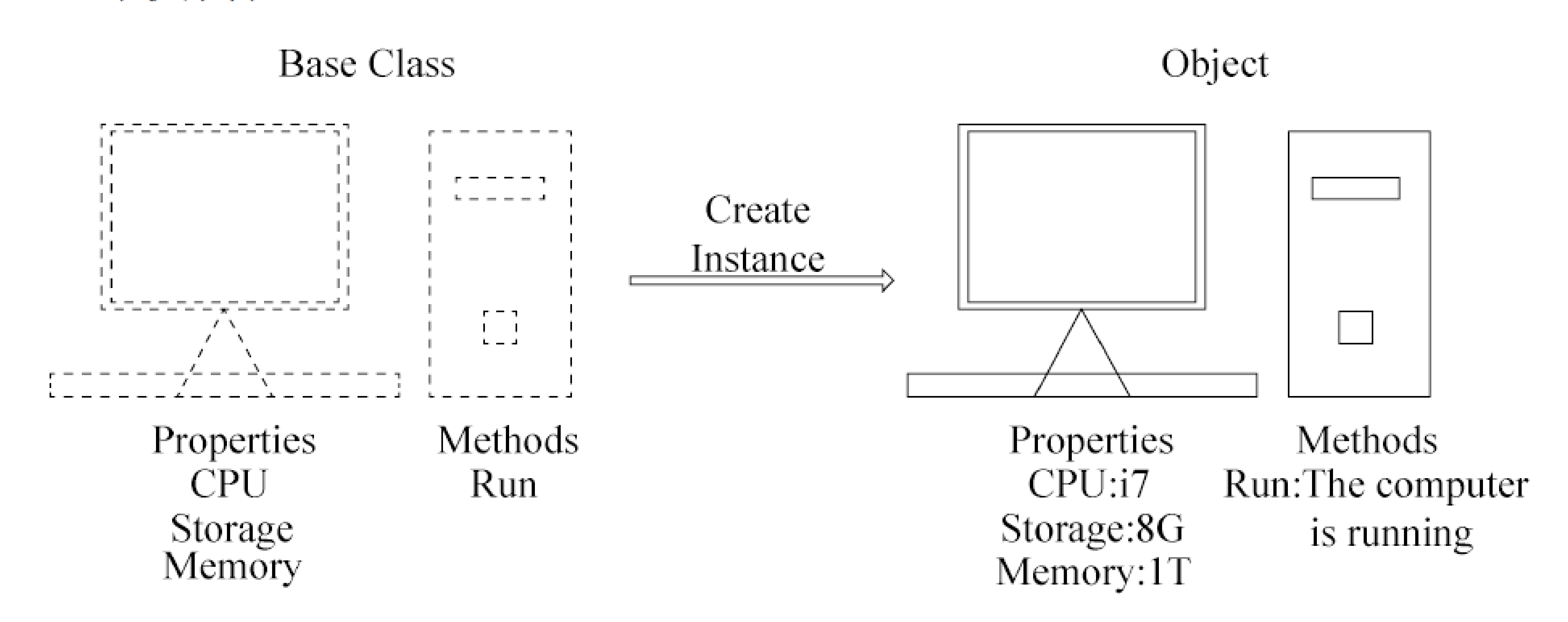


图 8.1　台式计算机类及其实例化

8.2.2　创建类和实例

上文提到过使用 class 语句创建类，使用类创建实例。接下来通过程序 8.1 来简单创建一个类和它的实例。

程序 8.1：

```
class machine:
    pass

```

```
4:    m = machine()
5:    print(m)
```

输出：

```
<__main__.machine object at 0x0031C4F0>
```

分析：

在 Python 中通过使用 class 语句与类的名称来创建一个新类，如程序的第 1 行，第 2 行是一个缩进的 pass 语句，在实际的开发中，这个地方代表类的主体，关于类的各种数据、功能都在这定义。完成类的定义之后，通过类的名称后跟一对括号的方法创建这个类的对象。

程序的输出表示 machine 类的__main__模块中拥有了一个实例。该实例对象存储在计算机地址为 0x0031C4F0 中（如果你在计算机中运行这段程序的话，输出的实例对象存储地址可能会不同）。

8.2.3　方法

现阶段我们可以简单地将类中的函数理解为方法。但是，在方法中我们还拥有一个额外的 self 变量。关于 self 和方法的详细操作如程序 8.2 所示。

程序 8.2：

```
1:    class machine:
2:        def printLine(self):
3:            print('hello\n')
4:
5:    m = machine()
6:    m.printLine()
```

输出：

```
hello
```

分析：

程序 8.2 的大部分和程序 8.1 相同，方法的使用如第 6 行所示，使用“.”来调用。我们再从整体上来看这个程序，在类中定义的 printLine 方法中有个 self 参数，但是，在使用实例调用的时候并没有加入 self 参数。这是因为 self 代表的是类的实例。在定义类的时候，

self 是必须有的，但在调用时不必传入与 self 相应的参数。

注意：

self 代表类的实例，而不是类。这也是类的方法与普通的函数的区别。

还记得程序 8.1 的输出吗？我们可以使用 self 来输出同样的结果，详细看看下面的这段程序：

```
class machine:
    def printLine(self):
        print(self)

m = machine()
m.printLine()
print(m)
```

上述程序会输出如下两行相同的内容：

```
<__main__.machine object at 0x0024C490>
<__main__.machine object at 0x0024C490>
```

上述程序明确地说明了 self 的内容，就是当前所在的对象。

8.2.4　__init__方法

在 Python 的类中，__init__方法具有特殊的意义。它主要是用于初始化实例的属性，该方法会在类的对象被实例化时立即运行，以此保证在此之后的方法调用时属性已存在，避免错误或异常发生。我们使用程序 8.3 来详细看看__init__的使用方法。

程序 8.3：

```
class machine:
    def __init__(self, type):
        self.type = type

    def ptr(self):
        print('This machine is {0}'.format(self.type))

m = machine('engine')
m.ptr()
```

输出：

```
This machine is engine
```

分析：

程序 8.3 中 machine 类中有两个方法，注意：类方法中必须要有 self，并且在调用这个功能时不必对这个参数进行赋值。接下来我们详细看看__init__方法的意义。

从程序输出可以看出，我们并没有调用__init__方法，但是输出时的确是输出了指定的 engine。这个正是__init__方法要做的。我们不必因为在第 3 行中使用两个 type 而担心混乱，这两个是不一样的变量，等号右边的 type 表示的是要传进来的值，它是一个局部变量。等号左边因为使用了 self.type，这意味着左边的 type 是 self 代表的也就是当前所在对象的一部分。

我们使用第 8 行的形式来创建了 machine 的一个实例 m，同时，Python 自动帮我们调用了__init__方法，将这行紧跟在类后的参数传给 self.type。

这时实例 m 中已经有了 type 这一属性，我们依旧是使用 self.type 来指定它，使用方法见程序的第 6 行。

8.2.5　类变量/对象变量

介绍完类的方法后，我们来看看类中的变量：属性，也可以理解为字段。它分为类变量和对象变量。其实这并不是什么稀奇古怪的知识。在上文的 machine 类中就已经出现过，machine 类中的 type 就是类的数据，具体地说，它是一个类变量。

再次详细查看程序 8.3 的第 3 行，其实属性只是将变量绑定到类和实例中，它们依托于类和实例，仅能通过类和实例使用。因此，类和实例的属性也称为绑定到类和对象命名空间的普通变量。

类和对象的数据部分有两种：类变量和对象变量。我们先来看看对象变量，它是由类的每个独立的实例所拥有，不会被共享。每个实例都拥有属于自己的属性副本。而对于类变量，它是被该类的所有实例所共享的。类变量只有一个副本，当任何一个对象对类变量做出改变时，发生的变动将更新到整个类的所有对象中。

接下来，通过程序 8.4 来看看类变量和对象变量的具体用法。

程序 8.4：

```
class Robot:
    num = 0
```

```

    def __init__(self, type):
        self.type = type
        Robot.num += 1
        print("Initializing {0} {1}".format(self.type, Robot.num))

    def work(self):
        '''simulated robot is working,wear and tear'''

        print("{0} Robot is working".format(self.type))

    def destroy(self):
        print("{0} Robot is being destroyed!".format(self.type))

        Robot.num -= 1

        if Robot.num == 0:
            print("robots are destroyed.")
        else:
            print("There are still {}".format(Robot.num))

    @classmethod
    def print_num(cls):
        print("There are {0} robots".format(cls.num))
#Robot class has finished

r1 = Robot("specialized robot")
r2 = Robot("industrial robot")

r1.work()
r2.work()
Robot.print_num()

r1.destroy()
r2.destroy()
Robot.print_num()
```

输出：

```
Initializing specialized robot 1
Initializing industrial robot 2
```

```
specialized robot Robot is working
industrial robot Robot is working
There are 2 robots
specialized robot Robot is being destroyed!
There are still 1
industrial robot Robot is being destroyed!
robots are destroyed.
There are 0 robots
```

分析：

在本程序中不仅可以看出类变量和对象变量的用法，还提到了类方法。程序创建了一个 Robot 类并定义了三个方法和一个类变量，最后通过调用方法来模拟机器人的创建、工作、销毁周期。

在程序的第 2 行声明的 num 是属于 Robot 类的一个类变量，它代表着实例化生产出机器人的总个数，而第 5 行初始化方法中使用 self 声明的 type 属于对象变量，它代表的是当前实例化生产机器人的类型。在使用它们时，类变量和对象变量的方法也不相同。我们通过 Robot.num（如程序的第 6 行）来指定类变量，通过 self.type（如程序的第 5 行）来指定对象变量，self 表示变量是当前实例所有的。

注意：

除了使用 Robot.num 的方法调用类变量还可以在类内使用 self.__class__.num，在类外使用 r1.__class__.num（r1 是 Robot 的对象）来调用类变量。

程序的第 25 行，这里的 print_num 和别的方法不同。实际上 print_num 是一个属于类而不属于对象的方法。由于它并不依附于具体的对象，在调用类方法的时候使用 Robot.print_num()的方式（程序的第 34 行），直接使用类名指定。

注意：

在使用类方法的时候一定要像程序 8.4 的第 24 行那样将方法标记为类方法，具体来说，标记类方法使用了一种名为装饰器的语法，启用@classmethod 等价于调用下面这条语句：print_num = classmethod(print_num)。

我们将程序的第 29 行和 30 行作为切入点，从总体上再次分析程序。

当 Robot 类声明完成后，用第 29 行的方法构建了一个 Robot 类的实例，并且参数 specialized robot 会传入初始化函数中，并使用它为对象变量 type 赋值。该方法的调用表示

有一个新的机器人被创建，因此，类变量 num 自增 1。

方法 work 虽然只是一条输出语句，我们用它来模拟机器人一直在工作。方法 destroy 用一条输出语句来模拟机器人被销毁的过程，在机器人被销毁后，对应类变量 num 减 1，之后使用 if 语句对当前的 num 值进行判断。

程序的第 30 行同 29 行相似，它们构造了 Robot 类的不同实例，对应着输出结果的第 3、4 行可以看出，对象变量是每个实例独有的。通过输出结果中机器人的数量随着 destroy 方法的调用而变换，我们可以看出类变量 num 是被所有 Robot 类的实例所共享的。

8.3　面向对象编程

在本章刚开始的时候，我们就提到面向对象的一个优点就是对代码的重用，而重用的一个重要实现方法就是通过继承机制来实现的。而多态、封装等技术就穿插在其中。

8.3.1　抽象

抽象是把一类事物的共有特征提取出来，它的本质在于提炼存在事物之间的共性。例如，对于人类我们可以抽象出姓名、年龄和性别这三个特性。而细化人类，可以继续分为中国人、美国人、英国人等，他们有着人类的特性，同时他们还有国籍、语言等特性。这个过程就是后文要提到的继承，关于这个过程的图解如图 8.2 所示。

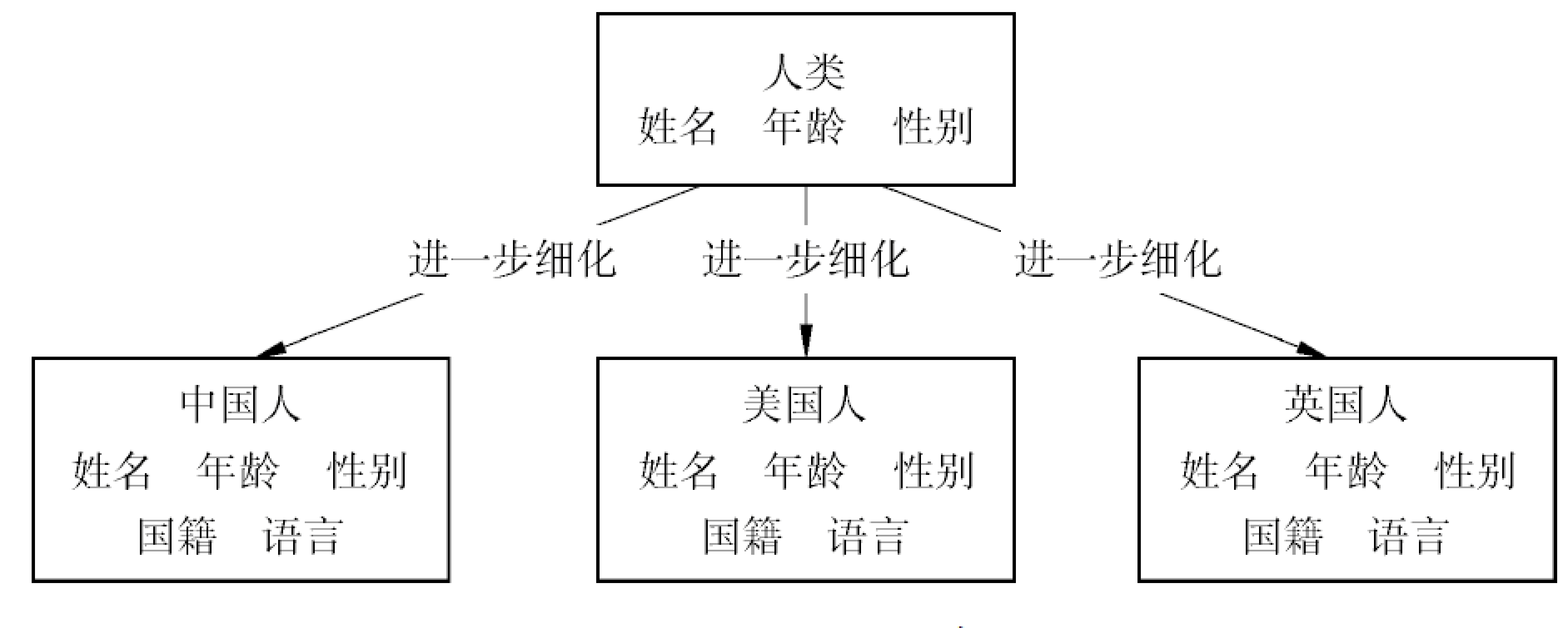


图 8.2　分类的细化过程

虽然在图 8.2 中第二级（中国人等）中仍使用浅颜色写了姓名等属性，但是实际上完全可以忽视它，因为我们可以从上一级人类中找到这些属性。通过这种形式，不仅节省了空间，还将问题的域或是复杂度降到了我们可控的范围。

在实际的编程中，我们往往会根据实际情况将问题划分为不同的层次，它是一个金字塔的结构，越往上，抽象度越高而且涉及的那些关于问题的细节越少。由于对问题的抽象是整个面向对象编程中的框架，因此如何抽象也是面向对象设计的难点。下面我们从一些简单的问题入手，简单讲解抽象的实现、继承机制及相关的技术。

8.3.2　继承和多态

在继承机制中，我们定义基类和派生类，可以将继承想象成父子关系。基类又称为父类、超类，派生类又被称为子类。具体地说，继承描述的是基类的内容如何传递给派生类，同时，派生类可以继承它基类的任何数据（属性和方法）。先通过下面的程序看看 Python 中继承的形式：

```
#基类
class Base():
        pass

#派生类
class Drived(Base):
       pass
```

其中，基类其实和普通的类在外形上没有什么不同，派生类也不过是在类名后用括号表明自己的基类（可以有多个基类）。关于继承及其相关技术，这一小段代码是不足以展现的，接下来我们来看一个较为完整的例子。

在人教版高中数学选修 1—1 中的第二章圆锥曲线（conic section）与方程中讲了椭圆、双曲线和抛物线。我们将圆锥曲线作为基类，选取椭圆（ellipse）和抛物线（para-curve）作为圆锥曲线的派生类，其中，圆锥曲线类中有一个属性和一个方法：曲线的名称和输出曲线的方程，派生类中是不同曲线的不同的参数（属性）和特有的方法。

对此，我们当然可以创建两个独立的类来分别表示这两种曲线，但是，若是想要添加一条共有的特征（如对曲线的焦点的操作），就意味着要对两个类都进行更新，这样会让程序越来越笨重，也会让开发者感到越来越无聊。因此我们将圆锥曲线作为基类提出，关

于类的具体设计如图 8.3 所示。

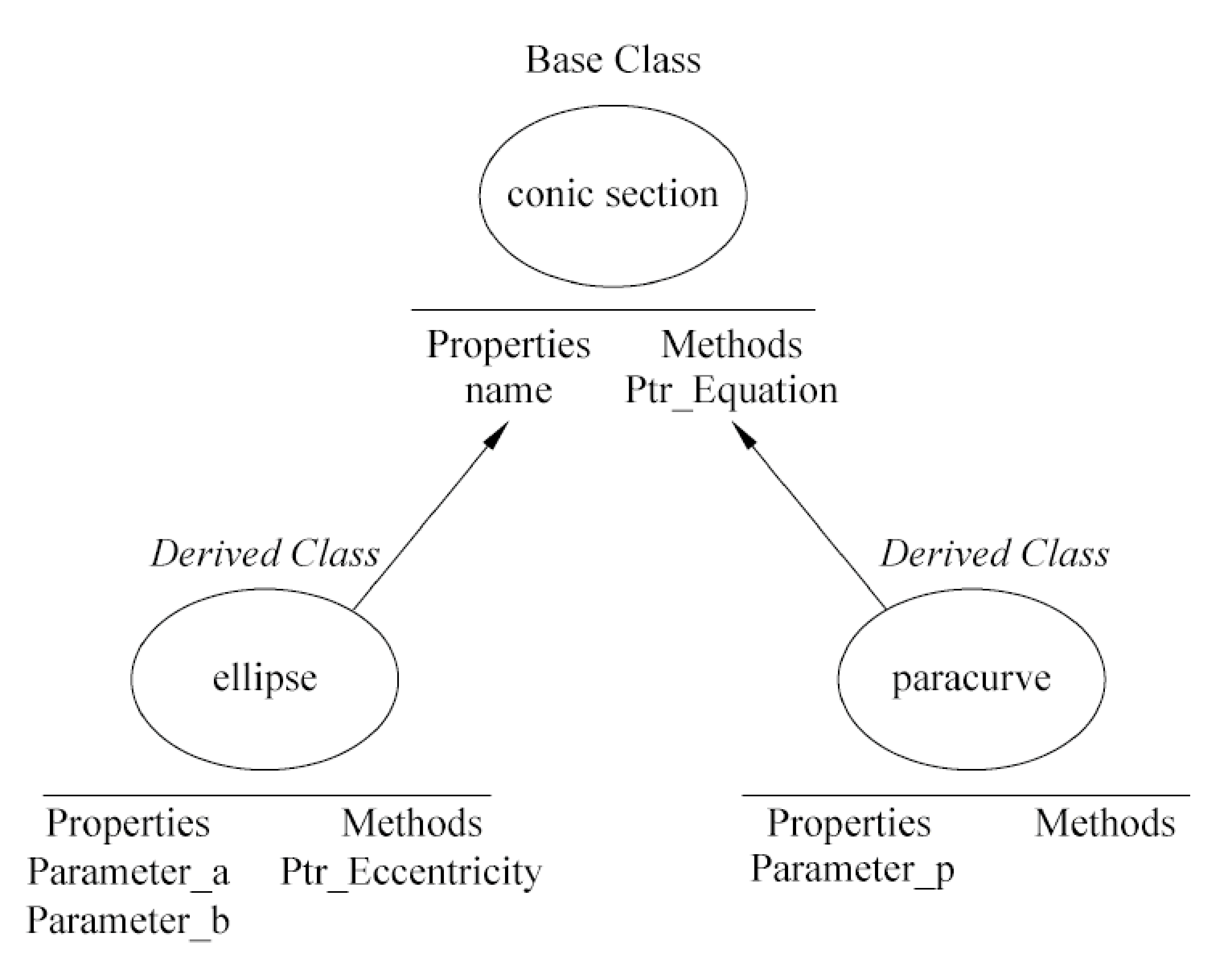


图 8.3　类的继承结构设计

通过图 8.3 的结构，修改曲线类时派生类中都会对应有所体现，同时，对派生类的修改仅仅会在当前类中体现。另外，派生类对象可以看作基类的对象，即派生类对象可以使用基类的属性或方法（如图 8.3 中所示，椭圆类的对象可以使用曲线类的 Ptr_Equation 方法输出），这个特点被称为多态性。

至此，是不是感受到了 Python 中的继承机制的优势，但是，不得不提的是派生类不仅可以使用继承自基类的方法，还可以覆盖掉基类的方法（例如，椭圆要输出一个通过属性中参数确定的具体椭圆方程）。曲线类及其继承类的具体程序如程序 8.5 所示。

程序 8.5　曲线类及其继承类的实现：

```
class conical_section:
    def __init__(self, name):
        self.name = name
        print("Determine the name of the curve: {0}".format(name))

    def ptr_equation(self):
        print("Output {0} equation".format(self.name))
```

```

class ellipse(conical_section):
    '''This class is used to simulate elliptic equation'''
    def __init__(self, name, a, b):
        conical_section.__init__(self, name)
        self.a = a
        self.b = b

    def ptr_eccentricity (self):
        print("the eccentricity of this ellipse is {0}".format((1-self.b**2/self.a**2)**0.5))

    def ptr_equation(self):
        conical_section.ptr_equation(self)
        print("x^2/{0}^2 + y^2/{0}^2 = 1".format(self.a, self.b))

class paracurve(conical_section):
    '''This class is used to simulate paracurve'''
    def __init__(self, name, p):
        conical_section.__init__(self, name)
        self.p = p

    def ptr_equation(self):
        conical_section.ptr_equation(self)
        print("y^2 = {0} * x".format(2*self.p))

e = ellipse("elliptic", 5, 3)
e.ptr_equation()
e.ptr_eccentricity()
print()
p = paracurve("para-curve", 7)
p.ptr_equation()
```

输出：

```
Determine the name of the curve: elliptic
Output elliptic equation
x^2/5^2 + y^2/5^2 = 1
the eccentricity of this ellipse is 0.8

Determine the name of the curve: para-curve
```

```
Output para-curve equation
y^2 = 14 * x
```

分析：

我们先来看看 ellipse 类，在定义这个类时类名后跟一个含有基类类名的元组，当然了元组中的基类不会只局限于一个。派生类 ellipse 中第 12 行通过类名显示调用基类的__init__方法，这里还有些细节要强调一下：调用基类__init__方法需要把派生类中的 self 作为参数传入。若是派生类中没有定义__init__方法，Python 会自动调用基类的__init__方法。

“派生类继承了基类的一切”这不是一句空话，我们使用同调用基类的__init__方法类似的手段来调用基类的其他方法，方法名前面加上基类名作为前缀，再将派生类的 self 以及其他参数传入（如程序的第 30 行）。

程序的第 34 行，当派生类调用类中与基类同名的方法时，调用的是派生类的方法。这是 Python 继承中的一种机制：Python 总是会从当前的实际类型中开始寻找方法，当在当前类中没找到时就会在该类所属的基类中依照顺序向上逐个寻找基类的方法，直到找到该方法。派生类定义与基类同名的方法并通过派生类调用该方法时会让基类方法隐藏，这种机制也被称为通过继承覆盖基类方法。

8.3.3　抽象类和抽象方法

抽象类和抽象方法和普通的类、方法没有太大的区别，存在抽象方法的类便是抽象类。抽象方法表示一个没有实现的方法，同时抽象类也不能被实例化。子类继承抽象类时一定要重写抽象方法，否则会报错。

有了上一小节的学习，我们对继承机制已经有了一定的了解，在这里我们直接取出程序 8.5 中的部分内容并将其改造为一个抽象类及其继承子类，如程序 8.6 所示。

程序 8.6　抽象类的实现：

```
from abc import ABCMeta,abstractmethod
class conical_section(metaclass=ABCMeta):
    def printInfo(self):
        print("this is a abstract class")

    @abstractmethod
    def ptr_equation(self):
        pass
```

```
9:
10:  class ellipse(conical_section):
11:      def ptr_equation(self):
12:          print("this is used to print ellipse's equation")
13:
14:  e = ellipse()
15:  e.printInfo()
16:  e.ptr_equation()
```

输出：

```
this is a abstract class
this is used to print ellipse's equation
```

分析：

程序的第 1 行将 ABC 模块的内容引入程序，这个模块为抽象类的实现提供支持，ABC 是 Abstract Base Class 的缩写。程序的第 2 行使用 metaclass=ABCMeta 将 conical_section 类声明为抽象类（没有这步声明 conical_section 类也是抽象类，因为类中有抽象方法），第 6 行使用装饰器将类中的 ptr_equation 方法声明为抽象方法，同时，方法体中使用 pass 表示无任何内容。

接下来使用 ellipse 类来继承抽象类，一旦选择继承抽象类那么就一定要重写对应的抽象方法。因此，程序的第 11 行对抽象方法重写，完成之后便可以通过子类实例调用具体的方法。

为什么要有抽象方法

学习完上述抽象的知识后，你是否觉得抽象方法的存在没有什么意义？貌似在程序 8.6 中我们绕了一圈却没多实现什么功能。但是，从整体的设计上讲，使用抽象类可以达到规范子类方法的作用，它在实际的开发中有着重要的意义。例如，我定义了一个汽车的类，它有启动、行驶、刹车、停止 4 个抽象方法。至于不同品牌、不同类型的汽车是钥匙启动还是一键启动，跑得多快，什么方式刹车这些不同的子类有不同的实现方式。但是，给子类自由发挥的空间的前提就是子类中一定要实现那 4 个抽象方法，否则就不是汽车了。

8.3.4　封装

下面来谈谈类的封装，顾名思义，封装就是将写在类中的某些信息对外隐藏。在 Python

中我们对想要特殊处理的变量，使用非常规的命名方式来告诉计算机这些变量要做特殊处理，先来看如何通过命名约定访问权限，如表 8.1 所示。

表 8.1　命名及其访问权限

序号	命名	对应权限
1	name	公开（public）
2	_name	语法上视为公开，但是内部约定含义为“可以被访问，但请视为私有，不要随意访问”
3	__name__	特殊属性
4	__name	私有（private），不能被继承类引用，不鼓励从外部访问

对于表 8.1 中的命名及其访问权限我们用到最多的是第 1 条和第 4 条。不过对于私有权限，你想明白为什么要使用它们了吗？从类本身的角度来说，使用私有权限最大的作用在于类可以管理自己的变量、方法。也就是说当你想访问类中内容时，类可以根据内容的敏感度选择对其保护。先来看下面的程序 8.7。

程序 8.7　封装：

```
class ellipse():
    '''This class is used to simulate elliptic equation'''
    def __init__(self, name, a, b):
        self.name = name
        self.__a = a
        self.__b = b

    def get_a(self):
        fake_a = self.__a + 1
        return fake_a

    def get_b(self):
        fake_b = self.__b + 1
        return fake_b

    def __ptr_eccentricity(self):
        print("the eccentricity of this ellipse is {0}"
          .format((1-self.__b**2/self.__a**2)**0.5))

    def ptr_equation(self):
```

```
20:         print("the ellipse: {}".format(self.name))
21:         print("x^2/{0}^2 + y^2/{0}^2 = 1".format(self.__a, self.__b))
22:
23: e = ellipse("elliptic", 5, 3)
24: print(e.get_a())
```

输出：

```
6
```

分析：

程序 8.7 的大部分内容截取自程序 8.5。因为对于一个椭圆来说，最重要的便是它的两个参数 a 和 b，在程序中的第 5、6 行，我们将参数设置为私有变量，对此，我们在类中添加两个返回参数的函数：get_a 和 get_b。通过观察这两个函数的内容我们也可以进一步理解封装的意义，没错，在函数中我们分别对参数 a，b 进行保护，最后将这个保护的参数作为真的参数传递出类。这也保证了类的安全性。

程序的第 16 行我们将输出曲线方程的方法也设置为私有的，因此使用类的实例不能对其进行访问。类中方法和变量的命名都遵从表 8.1 中的规则。

在这里还要强调一下，私有不允许通过实例对象直接访问，也不支持通过继承访问。

8.4　面向对象和面向过程的比较

学习到这里，面向对象部分已经讲完了。现在我们会明显地感觉到两者之间有着很大的区别。面向过程简单直接，易于入门理解，模块化程度较低。而面向对象相对于面向过程较为复杂，不易理解，模块化程度较高。关于两者比较的总结如下。

（1）两者都可以实现代码重用和模块化编程，但是面向对象的模块化层次更深，面向对象的封装性更强。

（2）面向对象的思维方式更加贴近于现实生活，更容易解决复杂的业务逻辑。从前期开发角度上来看，面向对象远比面向过程复杂，但是从维护和扩展功能的角度上来看，面向对象远比面向过程要简单。

（3）面向过程与面向对象的抉择，对于一个新手而言，从两者的对比就可以看出，当我们的业务逻辑比较简单时，使用面向过程能更快实现。前文中也提到过，Python 中面向

对象编程是可选的。但是当我们的业务逻辑比较复杂时，为了将来的维护和扩展，还是面向对象更为合适。

再使用一个通俗点的例子对比，面向过程编程就像一个缜密的流水线，从头到尾一气呵成，将所有的语句都在流水作业中执行。而面向对象编程就好像拼积木，我们将每个模块定义好，放在一旁，用到哪个就将其取来，最后堆积成一个完整的程序。

8.5　总　　结

如果只是通过了解面向对象的概念来学习本章知识可能会感到很吃力，本章的很多术语可能会令人感到困惑。在学习本章时，可以将概念和程序同时学习，双边相互参考并且将抽象的理论知识同现实的具体事物联想到一起。如果你之前接触过 C++或是 Java 的面向对象，那么本章讲的内容应该很平淡吧。若是之前没有接触过也没有关系，Python 将面向对象的很多复杂性质都去除了，我们可以将 Python 中的大部分内容简化成 Object. attribute 这一表达式。学到现在，我们对这个表达式已经不再陌生了，不管是方法还是属性，都是通过"."来调用的。关于面向对象我们先介绍到这里，但是关于面向对象的思考和练习一定不要到此为止，因为面向对象不仅是一门编程技术，还是一种经验，这种经验不被某种编程语言所限制。

8.6　练　　习

（1）通过添加子类内容来修改父类和直接修改父类内容哪一个更好？为什么？

（2）详细说说类和之前讲过的模块之间的关系。

（3）下面这段程序有哪些错误？

```
class ellipse():
    '''This class is used to simulate elliptic equation'''
def __init__(self, name, a, b):
        self.name = name
        self.a = a
        self.b = b
def __ptr_eccentricity(self):
```

```
print("the eccentricity of this ellipse is {0}"
        .format((1-self.__b**2/self.__a**2)**0.5))
def ptr_equation(self):
print("the ellipse: {}".format(self.name))
print("x^2/{0}^2 + y^2/{0}^2 = 1".format(self.__a, self.__b))

class ptr(ellipse):
    def __init__(self):
        ellipse.__init__("smaller ellipse", 3, 2)
    def __ptr_eccentricity(self):
        print("this is the ellipse equation")
    def ptr_parameter(self):
        print("The parameter of this ellipse is {0} and {1}".format(ellipse.a, ellipse.b)

p = ptr()
p.ptr_parameter()
```

披荆斩棘（选做）：

为你的学校编写一个人员管理程序，用户为教师和学生。他们共有的特征为姓名、年龄、住址以及假期。另外，教师独有特征：薪水、教学；学生独有特征：成绩、学费、上课。要求画出类设计图并编写程序（程序中的动作可以使用输出语句模拟）。

第 9 章　异　　常

学习到现在，我们编写的程序应该都或多或少地出现过异常，也许是不小心，也可能是逻辑错误或是程序以外的问题。不管是什么原因，异常是我们不希望看到的。在本章中，我们来看看关于异常的一些操作。通过本章学习，需要掌握：

- 什么是异常。
- 异常的用途。
- 异常的处理方法。
- 自定义异常的方法。

9.1　为什么要使用异常

异常会让我们从一个流程中跳出来，举个例子：工业制氨（人教版高中化学必修 1 最后一节），氨的工业制备是通过氮气和氢气在高温、高压加催化剂的条件下完成的。倘若这三个条件有一个不满足，化学反应就不会顺利进行。假设我们的设备都已经就绪，但是有条件不满足，可能是温度不够，可能压力达不到要求，或是催化剂失效，并不确定。但是设备都在准备着，我们需要快速地排除异常。若是不能快速排除异常，只能放弃此次制备，这会导致很大的损失。

上述情况正是异常要做的，我们可以在三个条件处添加异常，若是出现例外，终止制备过程进入异常管理器。在异常管理器中完成对当前异常的处理，不管是升温、加压还是更换催化剂，总之可以处理例外，让反应回到正轨。

9.2　异常的作用

通过上面工业制氨的例子，我们已经明白了什么是异常。但在程序中，异常的作用不

仅仅只是用于排除例外，接下来看看异常的具体作用。

1. 错误处理

在程序运行中发生错误时，Python 会向外抛出异常。Python 中有默认的异常处理机制，它会停止程序，打印出错信息。我们也可以使用 try 语句来捕捉异常并从异常中恢复。

2. 时间通知

异常也可以用于向外抛出程序的状态信息。例如，搜索程序在失败的时候返回一个异常信号。

3. 终止行为

使用 Python 的 try/finally 语句可以确保无论是否有异常都会执行 finally 中的语句。例如，在文件中搜索程序，无论搜索过程是否出现意外，都要执行 finally 语句中的关闭文件。

9.3　异常与错误

异常和错误是不一样的，下面通过程序 9.1 来介绍一下它们之间的区别。

程序 9.1：

```
Print("hello")

input("Please input something")
```

报错：

```
Traceback (most recent call last):
  File "/home/leo/PycharmProjects/error/error.py", line 1, in <module>
Print("hello")
NameError: name 'Print' is not defined
```

分析：

程序的第 1 行故意将 print 函数写成 Print（首字母大写），这样 Python 会抛出一个语法错误，为了方便我们纠错，Python 还会打印出检测到的错误发生的位置。接下来我们将第 1 行错误改正或是将第 1 行删掉，再次运行程序。这次的重点放在那条输入语句上，此时程序要尝试去读取输入的内容。我们按下 Ctrl+D 组合键，这表示一个文件结尾符号，但是它不应该在这里出现，所以 Python 报出了异常：

```
Traceback (most recent call last):
  File "/home/leo/PycharmProjects/hello/hello.py", line 3, in <module>
input("Please input something")
EOFError: EOF when reading a line
```

此处是指出了一个 EOFError 的错误，并且使用的是 Python 顶层默认的异常处理方式，它会立即终止程序。接下来我们来看看如何捕获异常并处理。在将具体的异常处理之前，让我们先理清概念，看看异常和错误的区别，如表 9.1 所示。

表 9.1　错误与异常比较

错误	异常
错误可分为语法错误和逻辑错误。 语法错误：代码不符合语言的语法。 逻辑错误：代码编写逻辑上的错误，程序不会报错但执行结果与预期不符	程序是可以运行的，但是在运行过程中遇到错误，会使程序意外退出

9.4　处 理 异 常

先来看看程序 9.2，它是程序 9.1 的扩展。

程序 9.2　异常处理：

```
try:
    str = input("Please input something: ")
except EOFError:
    print("why?")
else:
    print("right")

print("continue this program")
```

输出：

```
Please input something: ^D
why?
continue this program
```

分析：

程序中将有可能引发异常或是错误的语句（如程序的第 2 行）放在 try 中，except 后面紧跟异常类型（如程序的第 3 行的 EOFError），并在后续代码块中列出对上述异常类型做出的处理。如果没有提供异常类型，它将处理所有类型的异常。

当 try 语句用于捕获异常并从中恢复时，至少要有一句 except 与之对应，因为仅仅抛出异常是没有什么作用的。若是没有异常被处理，那么 Python 会调用默认处理器，它只会终端执行并打印出错误信息。

else 语句与 try…except 相关联，它的作用是在没有异常发生的时候执行某些操作。

接下来我们对应输出看看程序中语句的效果。输出的第 1 行对应的是程序中的 input 语句，其中“^D”表示的是按下了 Ctrl+D 组合键，这正好触发了一个 EOFError 类型的异常，转而进入程序的第 4 行的异常处理语句，完成处理后继续执行程序，进入程序的第 8 行输出 continue this program。

9.5　抛 出 异 常

Python 中有很多标准异常，常见的如表 9.2 所示。

表 9.2　Python 中常见的异常

序号	异常	解释
1	OverflowError	数值运算超出最大限制
2	ZeroDivisionError	除（或取模）零（所有数据类型）
3	OSError	操作系统错误
4	StopIteration	迭代器没有更多的值
5	FloatingPointError	浮点计算错误
6	IOError	输入/输出操作失败
7	MemoryError	内存溢出错误（对于 Python 解释器不是致命的）
8	EOFError	没有内建输入，到达 EOF 标记

但是，这些对于编程要面对的复杂环境来说还是不够的。因此，Python 允许我们自己定义一些异常类型并且通过 raise 语句来引发。我们要注意的是：自定义的异常必须是直接

或是间接从属于 Exception（异常）类的派生类。

接下来通过一个实验室制作氨气（人教版高中化学必修 1）的例子来了解一下如何自定义异常，实验如图 9.1 所示。

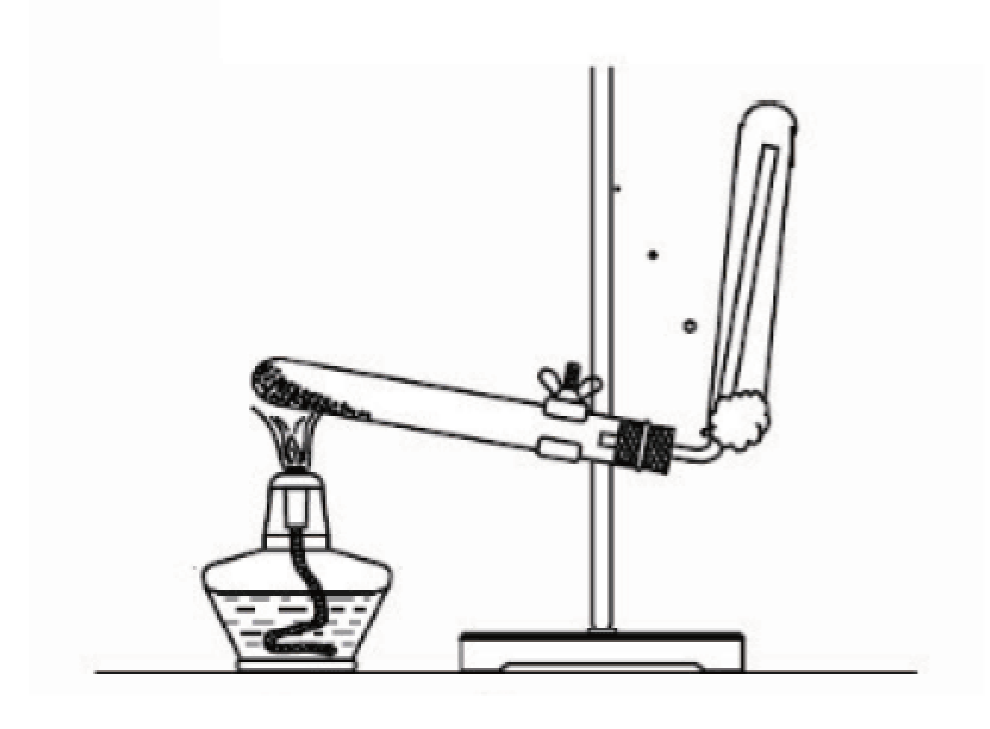


图 9.1　实验室制氨气

氨气的实验室制法是通过氯化铵晶体和消石灰固体经加热条件反应产生氨气，加热作为反应的重要条件，缺少它反应不会进行。现将是否加热条件放入异常，如果没有加热条件（如燃料不足等原因），程序抛出异常，检查酒精灯，最后刷洗实验器材，药品归位。具体程序如程序 9.3 所示。

程序 9.3　模拟无加热条件的实验室制氨气反应：

```
class NoHeating(Exception):
    '''Simulate the alcohol lamp does not work'''
    def __init__(self, state):
        Exception.__init__(self)

        self.state = state

#假设酒精灯处于燃油耗尽状态
state    = "exhausted"

try:
    #只有'y'代表准备好了
    if state != "y":
        raise NoHeating(state)
except NoHeating as nh:
    print("The alcohol lamp is {0},we are checking it.".format(state))
    #...
```

```
18:         #模拟正在添加酒精……
19:         #...
20:          print("It's done!")
21:    finally:
22:         print("Chemical reaction continue…\n")
23:    print("The experiment has been finished and returned the equipment.")
```

输出：

```
The alcohol lamp is exhausted,we are checking it.
It's done!
Chemical reaction continue…

The experiment has been finished and returned the equipment.
```

分析：

通过这个程序我们可以看出如何自定义异常。首先，程序的第 1 行中创建了我们自己的异常类，它是 Exception（异常）类的派生。它只接受一个状态信息并存储在类的 state 中。我们在程序中设定 state 为 exhausted（燃料用尽）以此来模拟一种异常情况。

在 try 的子句中，如果 state 不是'y'就抛出自定义的异常类 NoHeating。接下来在 except 子句中，我们将错误类 as（存储）成异常 nh 并在 except 子句中打印出 state 并对缺少燃料做出处理（此处并没有任何代码是关于添加燃料的，程序中仅仅是使用注释信息进行标注，位于程序的第 17～19 行）。处理完成后，输出完成提示信息。

最后，程序的第 21 行，我们会注意到这是一个很陌生的语句 finally，它表示无论 try 代码块中发生了什么，一定会在最后时执行收尾的代码。就像程序模拟的实验，不管程序有什么意外最后总会实验结束，归还器材。关于 finally 的一些具体事项我们在下一节详细介绍。

9.6　finally 语句

在上一节中，我们已经大致了解 finally 语句的作用。由于它一定会在最后执行的特点，finally 语句也被称为终止行为或是清理语句。我们用程序 9.4 详细谈谈 finally。

程序 9.4：

```
try:
    text = input("please input something : ")
```

```
finally:
        print("finally")
print("The last one")
```

输出 1：

```
please input something : dd
finally
The last one
```

输出 2：

```
please input something : ^D
Traceback (most recent call last):
  File "/home/leo/PycharmProjects/error/error_2.py", line 2, in <module>
text = input("please input something : ")
EOFError: EOF when reading a line
finally
```

分析：

对于程序 9.4 首先要强调的是程序没有使用我们常见的 try…except 的组合方式，而是使用了 try…finally 的组合方式。try…except 这种方式用于捕获异常并从中恢复，而对于 try…finally，可以确保无论 try 内的代码是否发生了异常，finally 中的终止行为一定会运行。例如，我们可以使用 try…except 来捕获读取文件过程中程序的异常并最后使用 try…finally 来确保文件关闭。当然了，也可以使用程序 9.3 那种 try…except…finally 的结构。

回到程序 9.4 中，我们对使用 input 输入来制造异常已经是非常熟悉了，重点看看输出 1 和输出 2。不管有没有异常，finally 都会被打印出来，而 print("The last one")这条语句却不是。

9.7　总　　结

异常是一种比较高级的控制流设备，它可能是由 Python 引发，也可能是由程序自身引发，在编程时要先分析并选取异常条件。我们在这里介绍的内容是比较基础的，要求大家熟练应用异常机制。

9.8　练　　习

（1）列出异常的几个优点。

（2）如何从异常中恢复程序？

（3）列出触发异常的几种方式。

披荆斩棘（选做）：

编写程序模拟工业制氨，要求将高温、高压、催化剂这三个条件作为不同的异常。

第 10 章　集合与概率

本章将学习 Python 中的集合数据类型并通过回顾概率的基本概念，了解其实际应用。通过本章的学习，需要掌握：

- ❑ 熟练使用 Python 中关于集合的操作。
- ❑ 了解概率的应用实例。
- ❑ 理解基于概率的朴素贝叶斯算法。

10.1　理解 Python 中的集合类型

Python set 是基本数据类型的一种集合类型，它与数学上的集合概念相同，在 Python 中集合分为可变集合（set）和不可变集合（frozenset）两种。以下将具体介绍集合的创建、添加、并集、交集、差集等实用操作。

以学校食堂每天购进水果的品种构成的集合为例，假设第一天购买水果的品种为苹果、草莓、香蕉、梨，第二天购买的品种为苹果、樱桃，那么请问：

（1）两天共买过多少样水果，其品种有哪些？

（2）两天所买相同水果的品种有哪些？

（3）两天中单独购买的水果的品种又有哪些？

我们以代码作答，具体程序如程序 10.1 所示。

程序 10.1　食堂进购水果案例：

```
first_day_food = set(['apple', 'strawberry', 'banana', 'pear'])
second_day_food = set()
second_day_food.add('apple')
second_day_food.add('cherry')
union_food = first_day_food.union(second_day_food)
print("{0} kinds of fruit were bought in two days, they are {1}"
        .format(len(union_food), union_food))
```

```
7:    the_same_food = first_day_food.intersection(second_day_food)
8:    print("{0} kinds of the same fruit were bought in two days, they are {1}"
            .format(len(the_same_food), the_same_food))
9:    first_day_diff_food = first_day_food.difference(second_day_food)
10:   second_day_diff_food = second_day_food.difference(first_day_food)
11:   print("The first day of the purchase of fruit are:{0}".format(first_day_diff_food))
12:   print("The second day of the purchase of fruit are:{0}".format(second_day_diff_food))
```

输出：

```
5 kinds of fruit were bought in two days, they are {'banana', 'strawberry', 'apple', 'pear', 'cherry'}
1 kinds of the same fruit were bought in two days, they are {'apple'}
The first day of the purchase of fruit are:{'pear', 'banana', 'strawberry'}
The second day of the purchase of fruit are:{'cherry'}
```

分析：

程序的第 1～4 行为构建集合与添加集合元素的代码，用于构建两天购买的水果清单，接着分别调用集合的 union、intersection 和 difference 函数，求取集合的并集、交集与差集。

通过上述例子，我们已经学会了集合的基本操作，而值得注意的是，集合是无序的、不重复的数据类型，因此不支持索引，不支持切片，也不支持重复。

我们可以用 Venn（文氏）图来表示程序 10.1。

首先，second_day 当中是空的，如图 10.1 所示。

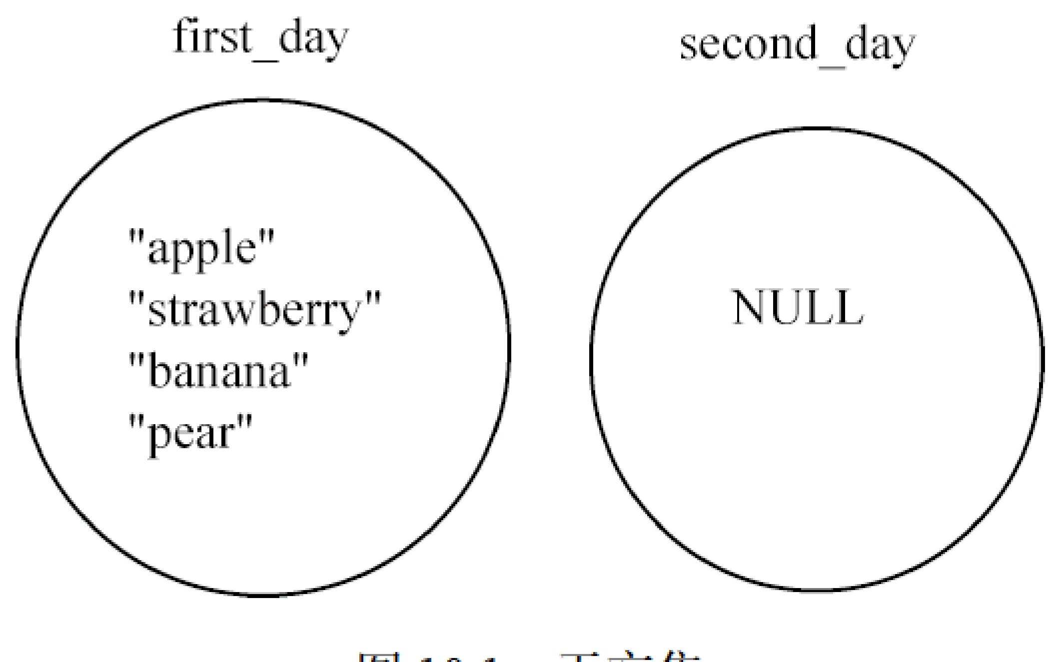


图 10.1　无交集

在 second_day 添加元素后，可以看出与 first_day 交集有 apple，如图 10.2 所示。

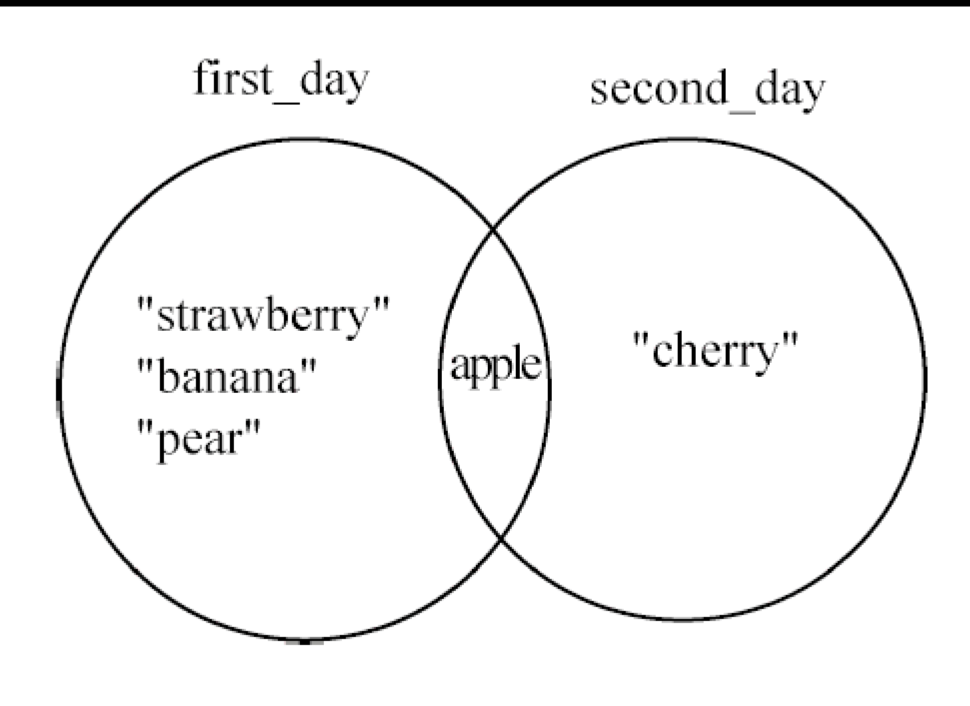


图 10.2　有交集 apple

通过图 10.1 和图 10.2 可以看出两个集合之间的关系。

10.2　概率基础知识

概率亦称“或然率”“几率”。它是反映随机事件出现的可能性大小的量度。随机事件是指在相同条件下，可能出现也可能不出现的事件。例如，在日常生活中，有些事件是必然发生的，有些是不可能发生的，但更多的事件发生的可能性是不确定的，对于不可能事件和必然事件，在试验中能够事先被确定是否会发生，我们称之为确定事件，而对于不确定事件（随机事件），其发生或不发生的可能性则称之为概率。

在概率中有着最为常见的两大概型：

（1）古典概型。古典概型讨论的对象局限于随机试验所有可能结果为有限个等可能的情形，即基本空间由有限个元素或基本事件组成，其个数记为 N，每个基本事件发生的可能性是相同的。若事件 A 包含 M 个基本事件，则定义事件 A 发生的概率为 $P(A)=\frac{M}{N}$，也就是事件 A 发生的概率等于事件 A 所包含的基本事件个数除以基本空间的基本事件的总个数，这是 P.S.拉普拉斯的古典概型定义，或称之为概率的古典定义。历史上古典概型是由研究诸如掷骰子一类赌博游戏中的问题引起的。计算古典概型，可以用穷举法列出所有基本事件，再数清一个事件所含的基本事件个数，然后相除，借助组合计算可以简化计算过程。

（2）几何概型。若随机试验中的基本事件有无穷多个，且每个基本事件发生是等可能的，这时就不能使用古典概型，于是产生了几何概型。几何概型的基本思想是把事件与几何区域对应，利用几何区域的度量来计算事件发生的概率，布丰投针问题是应用几何概型

的一个典型例子。

设某一事件 A（也是 S 中的某一区域），S 包含 A，它的量度大小为 $\mu(A)$，若以 $P(A)$ 表示事件 A 发生的概率，考虑到“均匀分布”性，事件 A 发生的概率取为 $P(A)=\mu(A)/\mu(S)$，这样计算的概率称为几何概型。若 Φ是不可能事件，即 Φ为 Ω中的空的区域，其量度大小为 0，故其概率 $P(\phi)=0$。

在介绍完概型后，我们介绍一下概率中最有名的大数定律。

大数定律（law of large numbers），是一种描述当试验次数很大时所呈现的概率性质的定律。但是注意到，大数定律并不是经验规律，而是在一些附加条件上经严格证明了的定理，它是一种自然规律，因而通常不叫定理而是大数“定律”。而我们说的大数定理通常是经数学家证明并以数学家名字命名的大数定理，如伯努利大数定理。简单来讲，它告诉我们在随机事件的大量重复出现中，往往呈现几乎必然的规律。在试验不变的条件下，重复试验多次，随机事件的概率近似于它出现的频率。

我们可以通过 Python 编程实验来证明大数定律，如程序 10.2 所示。

程序 10.2　大数定律：

```
import numpy as np
import matplotlib as mpl
mport matplotlib.pyplot as plt

mpl.rcParams['font.sans-serif']=[u'simHei']
mpl.rcParams['axes.unicode_minus']=False
# 定义数据量大小
numberSize = 200
# 生成服从正态分布的随机数据，其均值为 0
randData = np.random.normal(scale=100, size=numberSize)
# 保存随机每增加一个数据后算出来的均值
randData_average = []
randData_sum = 0
for index in range(len(randData)):
randData_average.append((randData[index] + randData_sum) / (index + 1.0))
# 定义作图的 x 值和 y 值
x = np.arange(0,numberSize,1)
y = randData_average
# 作图设置
plt.title('大数定律')
```

```
21:  plt.xlabel('数据量')
22:  plt.ylabel('平均值')
23:  # 画图并将图像进行展示
24:  plt.plot(x,y)
25:  plt.plot([0,numberSize], [0,0], 'r')
26:  plt.show()
```

输出，如图 10.3 所示：

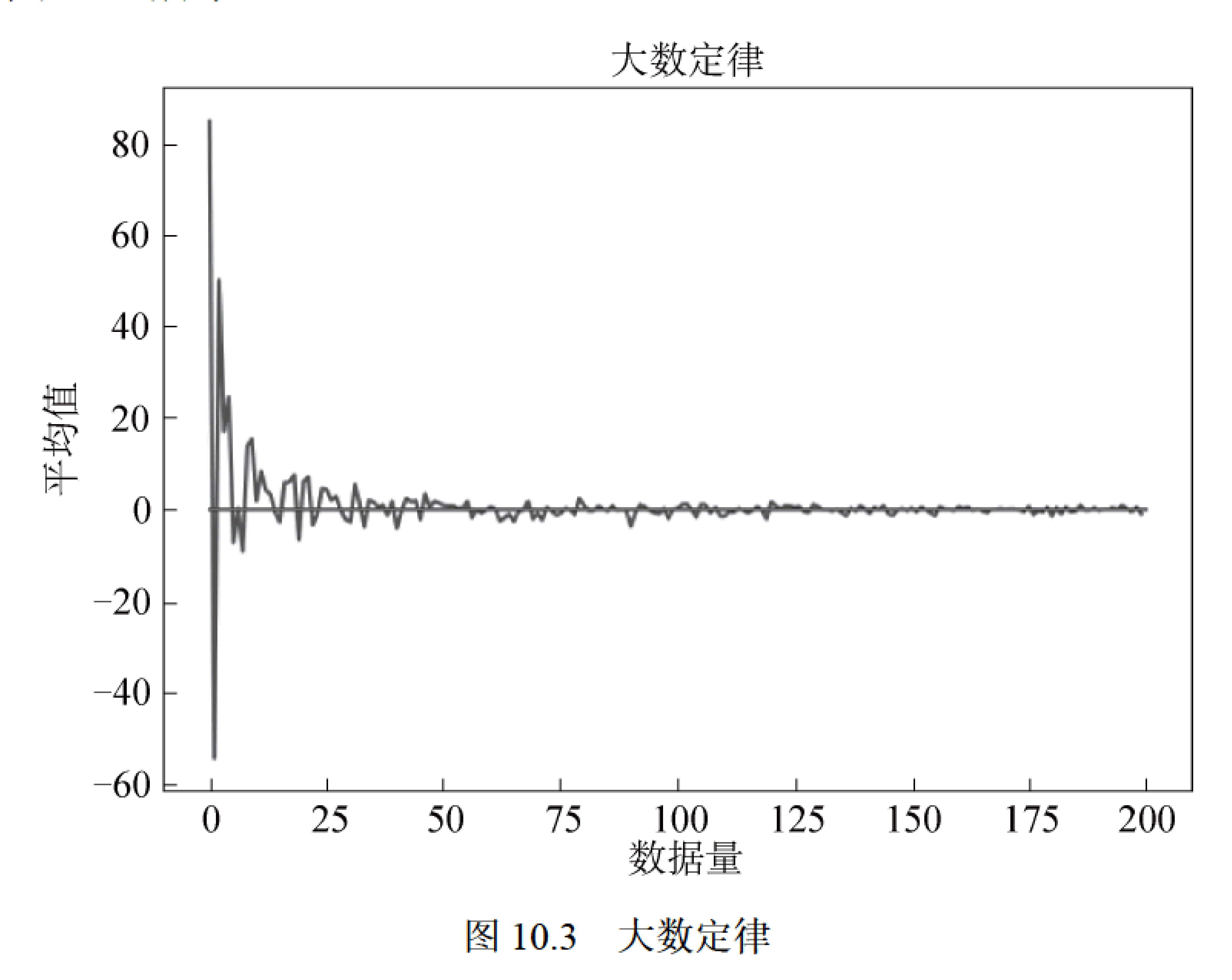


图 10.3　大数定律

分析：

程序的第 5 行指定默认字体，第 6 行则是防止由于在第 5 行更改了字体会导致不能将负号“–”显示出来，axes 相关设置里的设置是 axes.unicode_minus:True 代表默认情况采用的是 unicode 的 minus；所以在我们修改默认字体后需要将 True 改为 False。第 14 行代码通过循环计算每增加一个数据后的均值。

由图 10.3 即可进一步验证我们的结论，随着数据量的增加，均值越来越接近实际均值 0。即当我们的样本数据量足够大的时候，就可以用样本的平均值来估计总体平均值。

在日常生活中概率问题也十分常见，现举例如下。

例 1：根据生物学家的研究，人体的许多特征都是由基因控制的，有的人双眼皮，有

的人单眼皮，这是由一对人体基因控制的，控制单眼皮的基因 f 是隐性的，控制双眼皮的基因 F 是显性的，这样控制眼皮的一对基因可能是 FF 、 Ff 或 ff 。有基因 ff 的人是单眼皮的，有基因 FF 或 Ff 的人是双眼皮的。在遗传时，父母分别将他们所携带的一对基因中的一个基因遗传给子女，且可能性大小相等。如果父母都是双眼皮而他们的基因都是 Ff ，那么他们的子女双眼皮可能性大还是单眼皮可能性大呢？在这种情况下，我们只需将子女可能的基因组合都列举出来，求取各特征的概率即可，具体程序如程序 10.3 所示。

程序 10.3　基因组合：

```
1:  import itertools
2:  parents_gene = ["F", "f", "F", "f"]
3:  A = 0
4:  B = 0
5:  count = 0
6:  for ret in itertools.combinations(parents_gene, 2):
7:      if "F" in ret:
8:          B += 1
9:      else:
10:         A += 1
11:     count += 1
12:     prob_A = round(A / count, 2)
13:     prob_B = round(B / count, 2)
14: print("the probability of single eyelid is {0}".format(prob_A))
15: print("the probability of double-fold eyelid is {0}".format(prob_B))
```

输出：

```
the probability of single eyelid is 0.17
the probability of double-fold eyelid is 0.83
```

分析：

如程序的第 3、4 行所示，我们首先记子女单眼皮的事件为 A，双眼皮的事件为 B，并调用 itertools 模块的 combinations 方法将子女可能基因组合都列举出来，依次计数，由输出结果可知，子女的双眼皮的可能性较大。

提示：

这种通过列举法和图表法列举某事件，从而求取该事件概率的方法，我们称之为古典概型。

例 2：在如图 10.4 所示的边长为 2 的正方形中随机地撒一大把豆子，计算落在正方形里的圆中的豆子数与落在正方形中豆子数之比，并以此估计圆周率 π 。

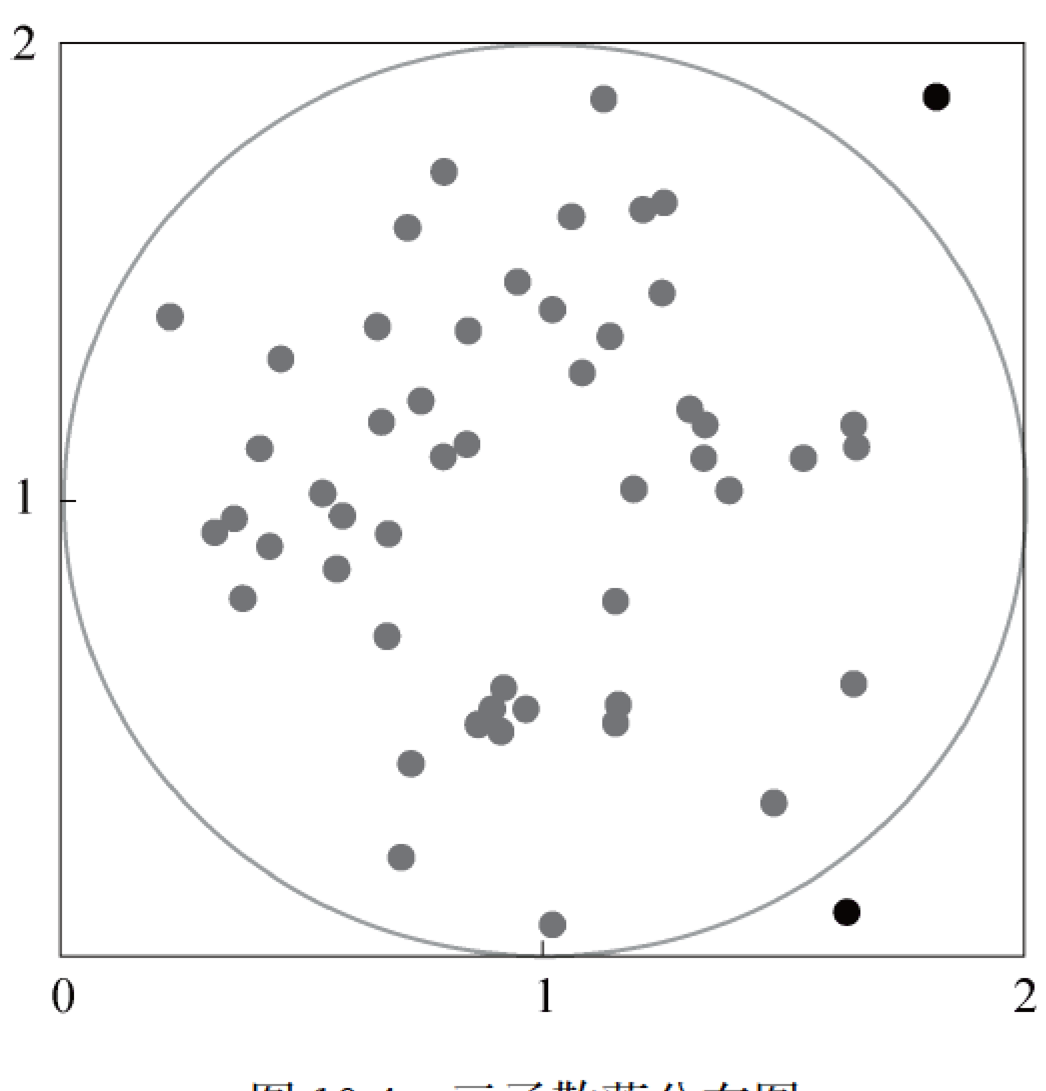


图 10.4　豆子散落分布图

如果我们把“在正方形中随机撒豆子”看成试验，把“豆子落在圆中”看成随机事件 *A*，则落在圆中豆子数与落在正方形中豆子数的比值就是事件 *A* 发生的可能性大小，如果撒的豆子数多的话，这时这个数据就可近似地看作事件 *A* 的概率，具体程序如程序 10.4 所示。

程序 10.4　近似计算圆周率：

```
import random
import math
n = 0
m = 100000
for i in range(m):
    x = random.random() * 2
    y = random.random() * 2
    if math.pow(x - 1, 2) + math.pow(y - 1, 2) <= 1:
        n += 1
pi_prob = n / m * 4
print("the pi is:{0}".format(pi_prob))
```

输出：

```
the pi is:3.13144
```

分析：

如程序第 4～9 行所示，我们在正方形中撒了 m 颗豆子，其中有 n 颗豆子落在圆中，根据面积比值 $\pi r^2/4r^2 = n/m$，圆周率 π 的近似值等于 $4n/m$。

提示：

这种概率的大小与面积的大小有关，即事件发生的概率等于满足组成图形的面积，这种概率模型我们可以称之为几何概型。

10.3　贝叶斯分类

贝叶斯分类算法在众多分类算法中占有重要地位，也属于统计学分类的范畴，是一种非规则的分类方法，贝叶斯分类技术通过对已分类的样本子集进行训练，学习归纳出分类函数（对离散变量的预测称作分类，对连续变量的分类称作回归），利用训练得到的分类器实现对未分类数据的分类。通过对比分析不同的分类算法，发现朴素贝叶斯分类算法（Naive Bayes），一种简单的贝叶斯分类算法，其应用效果比神经网络分类算法和判定树分类算法还要好，特别是当分类数据量非常大时，贝叶斯分类方法相较其他分类算法具有高准确率。

朴素贝叶斯分类是贝叶斯分类算法的一种，叫它朴素贝叶斯分类是因为这种方法的思想真的很朴素，朴素贝叶斯的思想基础是这样的：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。通俗来说，就好比你在街上看到一个黑人，如果让你猜他是从哪里来的，你十有八九会猜非洲。为什么呢？因为黑人中非洲人的比率最高，当然他也可能是美洲人或亚洲人，但在没有其他可用信息下，我们会选择条件概率最大的类别，这就是朴素贝叶斯的思想基础。

朴素贝叶斯分类的正式定义如下：

（1）设 $x=\{a_1,a_2,\cdots,a_m\}$ 为一个待分类项，而每个 a 为 x 的一个特征属性。

（2）有类别集合 $C=\{y_1,y_2,\cdots,y_n\}$。

（3）计算 $P(y_k\,|\,x),P(y_2\,|\,x),\cdots,P(y_n\,|\,x)$。

（4）如果 $P(y_k\,|\,x)=\max\{P(y_k\,|\,x),P(y_2\,|\,x),\cdots,P(y_n\,|\,x)\}$，则 $x\in y_k$。

贝叶斯分类的流程图如图 10.5 所示。

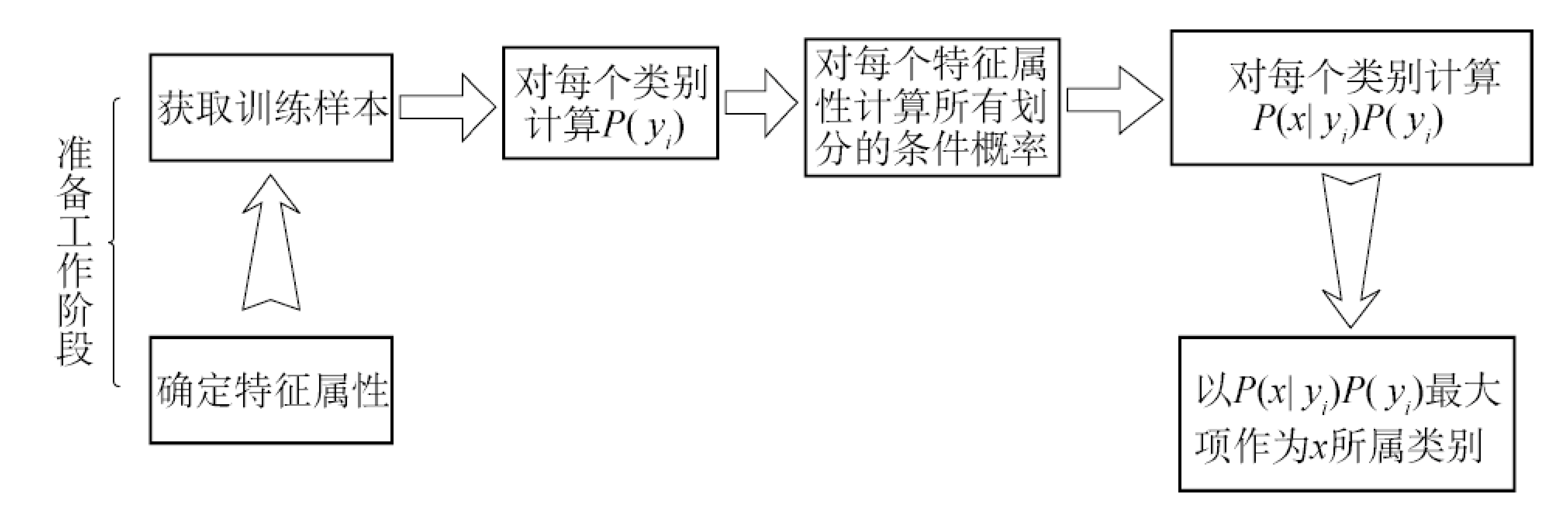


图 10.5　贝叶斯分类流程

下面我们利用贝叶斯分类来过滤垃圾邮件。

（1）我们用 *w* 表示邮件（E-mail）的特征向量（即 *w* 是一个向量），具体的一个特征表示某个单词是否出现过，1 表示出现，0 表示没有出现，此处单词的选取范围就是所有邮件出现过的单词。

假设我们的邮件如表 10.1 所示。

表 10.1　邮件短句

样本	E-mail 内容	标签
email　1	I love him	非垃圾邮件
email　2	You are a pig	垃圾邮件
email　3	I will go to school	非垃圾邮件
email　4	You are a dog	垃圾邮件

所有样本中出现过的单词的集合：[a,are,dog,go,him,i,love,pig,school,to,will,you]，单词会转化为小写，其中如 a,i 比较简短的单词（这个单词长度自己设置，还有比如语气助词单词，它们对邮件分类没有什么作用，可以考虑过滤掉）过滤后如下：[are,dog,go,him,love,pig,school,to,will,you]，对应于上面各样本的 *w* 如表 10.2 所示。

表 10.2　关键字转换向量

E-mail	*w*	*c*
email　1	[0,0,0,1,1,0,0,0,0,0]	0
email　2	[1,0,0,0,0,1,0,0,0,1]	1

续表

E-mail	w	c
email　3	[0,0,1,0,0,0,1,1,1,0]	0
email　4	[0,1,0,0,0,0,0,0,0,1]	1

（2）c 表示垃圾邮件的类别，此处分为垃圾邮件和非垃圾邮件两个类别，1 表示垃圾邮件，0 表示非垃圾邮件。

（3）计算 $P(c)$，我们可以通过统计样本的标签，就可以计算出垃圾邮件、非垃圾邮件出现的概率，如表 10.3 所示。

表 10.3　垃圾邮件与非垃圾邮件出现概率

标签	出现次数	概率 $P(c)$
0	2	0.5
1	2	0.5

（4）计算 $P(w|c)$，首先将邮件通过标签分开，即垃圾邮件部分和非垃圾邮件部分，然后统计单词在各个类别中出现的概率，即 $P(w|c)$，如表 10.4 所示。

表 10.4　单词出现频率

非垃圾邮件	email　1	[0,0,0,1,1,0,0,0,0,0]	2
	email　3	[0,0,1,0,0,0,1,1,1,0]	4
	求和	[0,0,1,1,1,0,1,1,1,0]	6
	$P(w\|c)$	[0,0,1/6,1/6,1/6,0,1/6,1/6,1/6,0]	
垃圾邮件	email　2	[1,0,0,0,0,0,0,0,0,1]	3
	email　4	[0,1,0,0,0,0,0,0,0,1]	2
	求和	[1,1,0,0,0,1,0,0,0,2]	5
	$P(w\|c)$	[1/5,1/5,0,0,0,1/5,0,0,0,2/5]	

通过表 10.4 所得的数据，我们开始进行预测分类，对于一个具体样本，其特征假如为 w，通过 $P(w)$和 $P(w|c)$来确定属于哪一个。

假设测试邮件为：pig dog,fuck dog，比较得出表 10.5。结果证明这是封垃圾邮件。

表 10.5　垃圾邮件

$P(w_i \mid c_i)$	$P(w_i \mid c_0)$	$P(w_i \mid c_1)$
	[0,0,1/6,1/6,1/6,0,1/6,1/6,1/6,0]	[1/5,1/5,0,0,0,1/5,0,0,0,2/5]
$P(c_i \mid w)$	$1 \cdot 0 \cdot P(c_0)=0$	$1 \cdot 1/5 \cdot P(c_1)=1/10$

10.4　案例：线上课程分类

在人工智能领域中，概率的应用是非常常见的，如上面介绍的垃圾邮件过滤中的贝叶斯分类算法、网店中的协同推荐算法等，下面我们将以课程标题分类为例，具体介绍基于条件概率的朴素贝叶斯算法：网站管理员最近在整理某教学网站上的英语材料，为提高工作效率，需要计算机帮他自动归类材料，于是他统计了该网站近期发布的课程标题，如表 10.6 所示。

表 10.6　课程标题表

序号	课程标题	类别
1	Daily English Learning	1
2	Welcome! - VOA Learning English	1
3	World of Math Online	0
4	Free online math lessons	0
5	Listen and Learn English	1
6	The physical learning style	0
7	Learning by Choice in Secondary Physical Education	0
8	Learn English in 30 Minutes	1

观察表 10.6 可以发现，在每个标题中都会有一两个关键字决定着标题类别，于是他通过统计每个单词在特定类别下的词频，尝试发现词频与类别的关联关系，其词频统计图如图 10.6 所示。

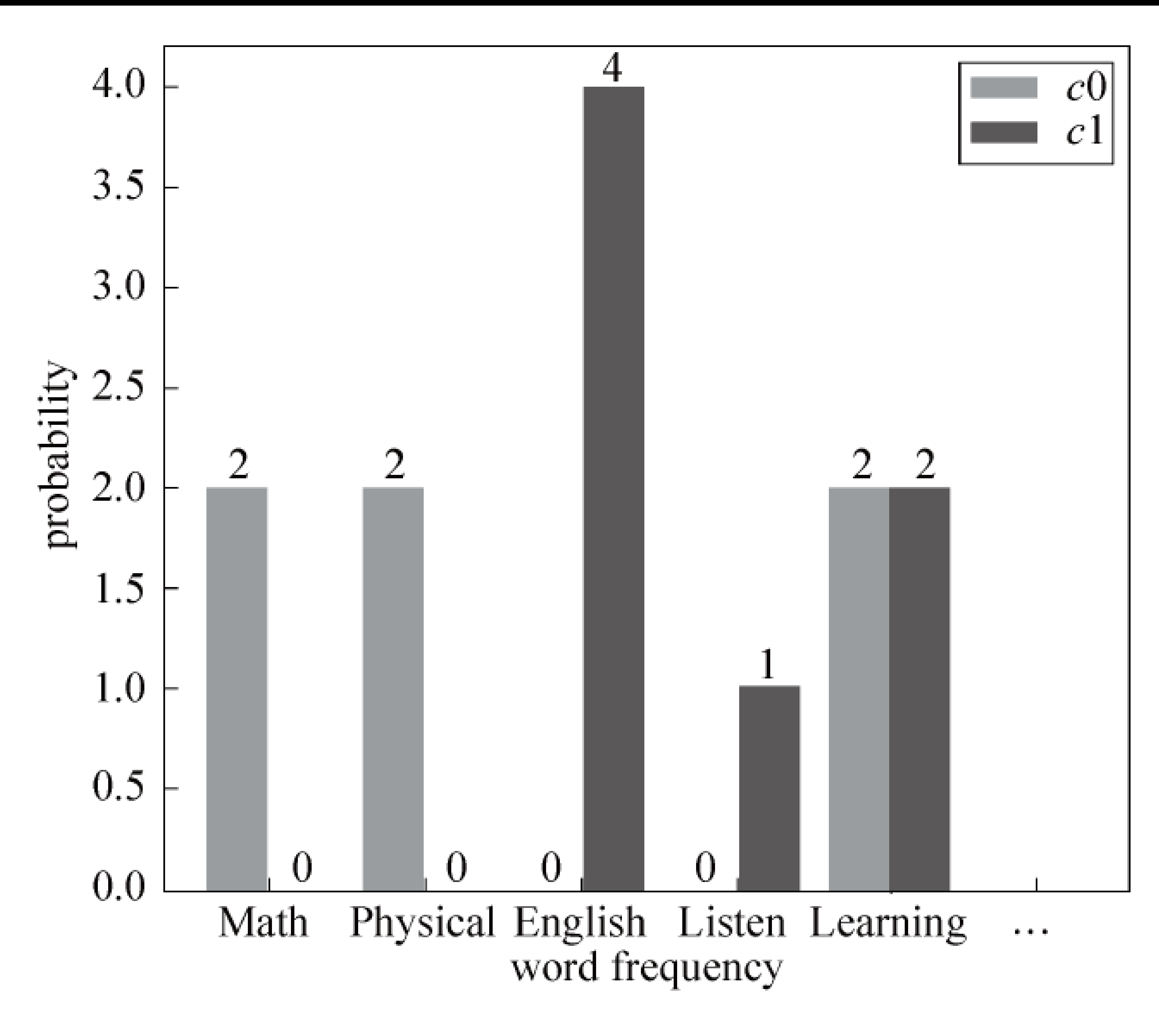


图 10.6　词频统计图

由图 10.6 可知，在英语材料的标题中，关键字 Listen、English 出现频率较高，而在其他类别课程的标题中则相对较低，因此只要通过比较单词出现在类别中的词频大小，即可对待分类标题进行归类。

（1）贝叶斯公式。在上述问题中，我们得到了每个单词在特定类别下的词频，记单词向量 $(x_1,x_2,\cdots,x_n)$ 为特征变量 X，其词频可表示为 $P(X|c_j)$，根据贝叶斯公式，我们可以计算在特征变量 X 出现的条件下，课程标题属于类别 c_j 的概率 $P(c_j|X)$，如公式（10-1）所示。

$$P(c_j|X)=\frac{P(X|c_j)P(c_j)}{P(X)} \tag{10-1}$$

其中，$P(c_i)$ 表示标题类别的概率，$P(X)$ 表示特征变量 X，即单词在所有标题中出现的概率。

（2）朴素贝叶斯公式。现在我们假设单词间是互相独立的，且对于每个 $P(X|c_j)$ 的计算 $P(X)$ 都一样，因此课题类别 c_j 的条件概率可简写为：

$$P(c_j|X)=\prod_{i=1}^{n}P(x_i|c_j)P(c_j) \tag{10-2}$$

在理解了上述概念后，我们打开文本编辑器，创建名为 naive_bayes.py 文件，并在 naive_bayes.py 文件中创建 Naive_bayes 类，如程序 10.5 所示。

程序 10.5　创建 Naive_bayes 类：

```
class Naive_bayes(object):
    def __init__(self, p0_vector, p1_vector, p_absolute, vocab_set):
        """
        输入：朴素贝叶斯实例 self，p0_vector 表示类别 c0 的单词词频 P(X | c0)，
        p1_vector 表示类别 c1 的单词词频 P(X | c1)，p_absolute 表示类别 c1 的概率
        P(c1)，单词字典 vocab_set
        输出：无
        描述：朴素贝叶斯构造函数
        """
        self.p0_vector = p0_vector
        self.p1_vector = p1_vector
        self.p_absolute = p_absolute
        self.vocab_set = vocab_set
        super(Naive_bayes, self).__init__()
```

在 Naive_bayes 类计算条件概率 $P(x_i \mid c_j)$ 前，需要做一些准备工作，首先我们基于训练样本构建单词字典，然后根据该字典，为每个样本生成对应的文本向量，其代码如程序 10.6 所示。

程序 10.6　构建文本向量：

```
def create_vocab_list(self, dataset):
    """
    输入：朴素贝叶斯实例 self，训练数据集 dataset
    输出：无
    描述：根据测试样本构建单词字典
    """
    vocab_set = set([])
    for document in dataset:
        vocab_set = vocab_set | set(document)
    self.vocab_set = list(vocab_set)

def wordset2vector(self, inputset):
    """
    输入：朴素贝叶斯实例 self，单条文本 inputset
    输出：文本向量 return_vec
```

```
        描述：将每条文本转换为数字向量，建立与字典同等大小的文本向量，若语句
        中的单词在字典表中出现则标记为 1，否则为 0
        """
        return_vec = [0] * len(self.vocab_set)
        for word in inputset:
            if word in self.vocab_set:
                return_vec[self.vocab_set.index(word)] += 1
        return return_vec
```

我们以课程标题为例，随机选择一个新标题 title = Listen English，构建特征变量 $X=(\text{Listen}, \text{English})$，根据表 10.6 的信息可统计出单词在特定类别下的词数，如表 10.7 所示。

表 10.7　单词词数统计表

X	c_0	c_1
Math	2	0
Physical	2	0
English	0	4
Listen	0	1
Learning	2	2
…	…	…

由此我们可以计算出测试样本中每个单词的条件概率 $P(x_i \mid c_j)$，如 $P(x=\text{Math} \mid c_0)=2/19=0.11$，$P(x=\text{Math} \mid c_1)=0/19=0$。接着计算类别 c_j 的概率，分别为 $P(c_0)=4/8=0.5$，$P(c_1)=4/8=0.5$，最后将上述概率依次代入公式（10-2），计算课题类别 c_j 的条件概率 $P(c_j \mid X)$：

$$P(c_0 \mid X)=P(\text{Listen} \mid c_0)\bullet P(\text{English} \mid c_0)\bullet P(c_0)=\frac{0}{19}\times\frac{0}{19}\times 0.5=0$$

$$P(c_1 \mid X)=P(\text{Listen} \mid c_1)\bullet P(\text{English} \mid c_1)\bullet P(c_1)=\frac{1}{15}\times\frac{4}{15}\times 0.5=0.009$$

其中，我们发现一个问题，Listen 和 Englsih 并未出现在任何非英语课程的标题样本中，导致与它相关的课题类别 c_j 的条件概率 $P(c_j \mid X)$ 一直等于 0，得不出任何信息。为避免此问题，我们需要使用拉普拉斯平滑方法：首先为特征变量 X 每个单词计数加 1，使它不会

为零。然后将外加的词数加到除数，这样计算的概率将永远不会大于 1。现在我们再次计算 $P(c_j \mid X)$：

$$P(c_0 \mid X) = P(\text{Listen} \mid c_0) \bullet P(\text{English} \mid c_0) \bullet P(c_0) = \frac{1}{19+2} \times \frac{1}{19+2} \times 0.5 = 0.001$$

$$P(c_1 \mid X) = P(\text{Listen} \mid c_1) \bullet P(\text{English} \mid c_1) \bullet P(c_1) = \frac{1+1}{15+2} \times \frac{4+1}{15+2} \times 0.5 = 0.017$$

因为 $P(c_0 \mid X) < P(c_1 \mid X)$，最后我们可以把标题 title = Listen English 划分到英语教学类中。其中，完整朴素贝叶斯分类器的训练方法如程序 10.7 所示。

程序 10.7　构建朴素贝叶斯分类器：

```
def compute_condition_probability(self, words_vec, labels):
    """
    输入：朴素贝叶斯实例 self，训练文本向量集合 words_vec，文本标签 labels
    输出：无
    描述：根据文本向量集合计算类别 c_i 的单词词频 P(X | c_i) 和概率 P(c_1)
    """
    num_train_docs = len(words_vec)   #训练样本数
    num_words = len(words_vec[0])     #训练样本中的单词总数
    p0_num = np.ones(num_words)       #拉普拉斯平滑
    p1_num = np.ones(num_words)
    for i in range(num_train_docs):
        if labels[i] == 1:
            p1_num += words_vec[i]
        else:
            p0_num += words_vec [i]
    self.p0_vector = np.log(p0_num / sum(p0_num)) #调用 log 函数，防止计算结果下溢
    self.p1_vector = np.log(p1_num / sum(p1_num))
    self.p_absoluate = sum(labels) / float(num_train_docs)

def fit(self, dataset, labels):
    """
    输入：朴素贝叶斯实例 self，训练样本集合 dataset，文本标签 labels
    输出：无
    描述：根据训练样本集训练朴素贝叶斯分类器
    """
    self.create_vocab_list(dataset) #构建样本单词字典
    words_vec = []
    for inputset in dataset:
```

```
        words_vec.append(self.wordset2vec(inputset))#构建文本向量
    self.compute_condition_probability(words_vec, labels) #计算条件概率 P(X | c_i) 和类别 c_1 的概
    率 P(c_1)
```

在训练完分类器模型后，我们再使用一个简单的概率比较即可完成对象的自动归类，其代码如程序 10.8 所示。

程序 10.8　构建朴素贝叶斯分类器：

```
def predict(self, word_vec):
    """
    输入：朴素贝叶斯实例 self，测试文本向量 word_vec
    输出：word_vec 所属类别
    描述：利用朴素贝叶斯分类器预测文本类别
    """
    p0 = sum(word_vec* self.p0V) + np.log(1.0 - self.p_absoluate)
    p1 = sum(word_vec* self.p1V) + np.log(self. p_absoluate)
    return 1 if p1 > p0 else 0
```

至此，让我们回顾上述算法步骤，可以整理出一个完整的朴素贝叶斯算法流程，如图 10.7 所示。

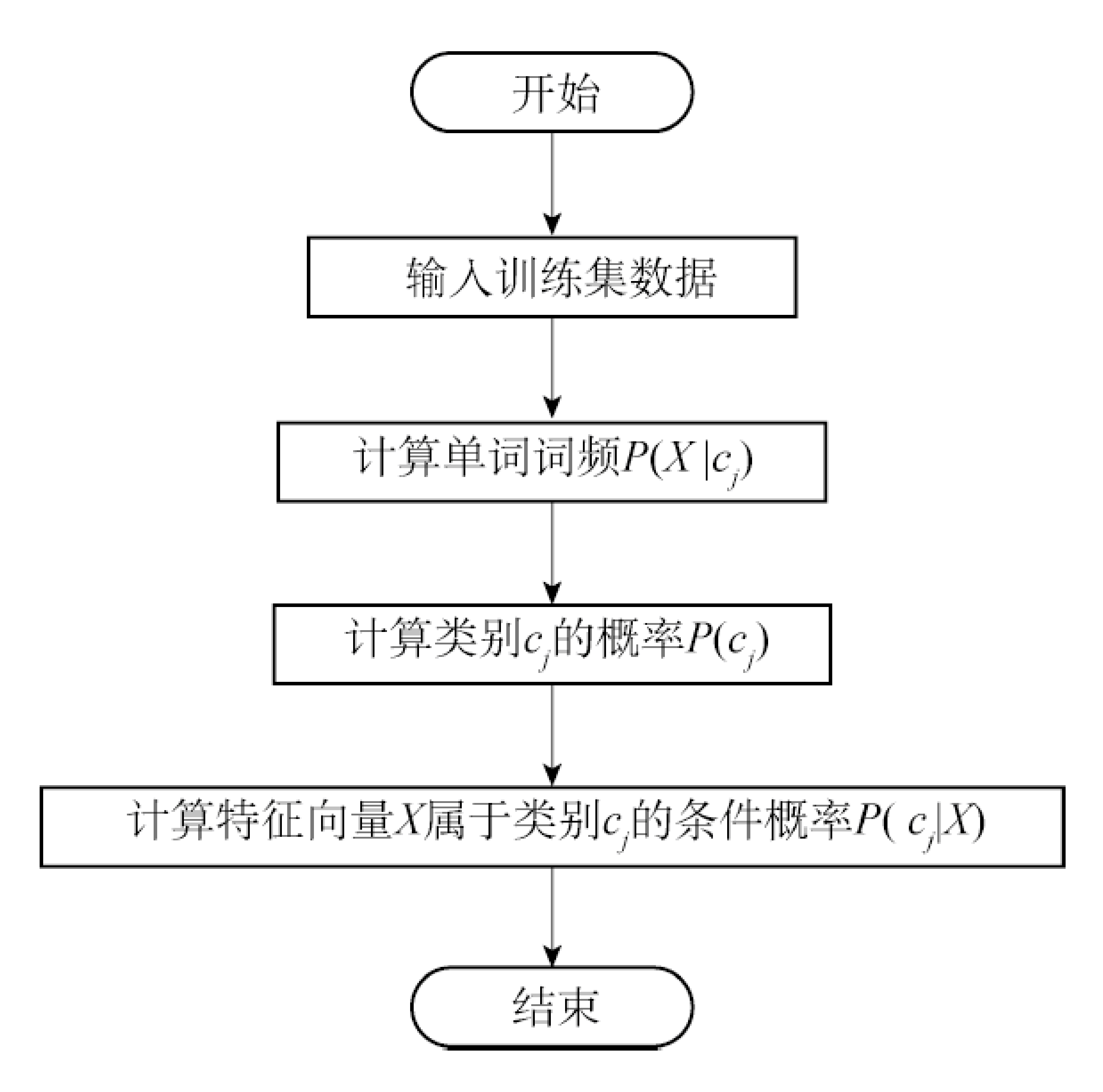


图 10.7　朴素贝叶斯算法流程图

由图 10.7 可知，朴素贝叶斯模型的构建是一个简单的线性流程，我们只需依次计算训练样本的单词词频 $P(X|c_j)$，样本类别概率 $P(c_j)$，进而求解特征向量 X 属于类别 c_j 的条件概率 $P(c_j|X)$，即可结束算法的训练过程。

到现在为止，我们已经大致完成了对朴素贝叶斯算法的学习，现在只需要通过一个数据加载和入口函数就可得到一个可运行的代码实例，其代码如程序 10.9 所示。

程序 10.9　数据加载函数和入口函数：

```
def load_dataset(filename, delimiter=" "):
    """
    输入：数据文件路径，分隔符
    输出：数据集
    描述：读取数据文件生成 np.nArray 类型的数据集
    """
    dataset = []
    labels = []
    with open(filename, 'r') as fp: #数据文件内容格式"Daily English Learning 1"
        while True:
            lines = fp.readline().strip() #lines = "Daily English Learning 1"
            if not lines:
                break
            feature = lines.split(delimiter)# feature = ['daily', 'english', 'learning', '1']
            key = int(feature [-1])
            values = [v.lower() for v in feature [0:-1]]
            labels.append(key)
            dataset.append(values)
    return dataset, label

if __name__ == "__main__":
    filename = "bayes.data"
    dataset, labels = load_dataset(filename)
    naive_bayes = Naive_bayes()
    naive_bayes.fit(dataset, labels)
    testset = ['learning', 'english']
    test_vec = naive_bayes.wordset2vec(testset)
    estimate = naive_bayes.predict(test_vec)
    print('[{0},{1}] estimate is: {2}'.format (testset[0], testset[1], estimate))
```

10.5 总　　结

本章我们分别回顾与学习了集合和概率的基本概念及其在实际中的应用，如集合的并集、交集，概率的古典概型与几何概型等，并通过线上课程分类的案例理解了基于概率的朴素贝叶斯分类算法工作流程与详细步骤。

10.6 练　　习

（1）定义一个函数 CreateNum()，返回一个列表，列表中包括 100 个数字，将小于 100 的，小于 500 大于 100 的，大于 500 的分别存入三个不同的集合当中。

（2）定义一个函数 dice(num，point，count)模拟掷骰子，其中 num 表示抛掷的骰子数，point 表示抛掷的点数，count 表示抛掷点数为 point 的次数，要求在函数中以重复试验的方式求出抛掷 num 个骰子，至少有 count 个点数为 point 的概率。

（3）使用贝叶斯分类器练习垃圾邮件过滤，两份邮件信息分别为 I LOVE YOU 与 YOU ARE A PIG。

第 11 章　学点统计学

本章将通过回顾统计学中的基本概念及现实意义，并根据实际案例介绍基于 Python 的统计回归模型，通过本章的学习，需要掌握：

- 加深对统计学相关概念的理解与应用。
- 理解假设检验的基本概念与计算步骤。
- 理解方差分析的概念与计算过程。
- 理解统计回归中线性回归模型。

11.1　统计学的基本概念

统计学是研究随机现象中确定的统计规律的学科，主要体现在收集、整理、分析与解释统计数据，并对其所反映的问题的性质、程度和原因做出一定结论。其计量资料的统计描述主要如下。

（1）均值：标志资料总量与单位总数的比值，计算简单，但易受极值影响，如程序 11.1 所示。

程序 11.1　计算均值：

```
List=[1,2,3,4,5,6,7,8,9]
print(sum(List)/len(List))
```

输出：

```
5.0
```

分析：

sum()为求和函数，len()可以计算列表长度。

（2）中位数：由小到大排列居中间位置的标志值，不易受极值影响，稳定性高，如程序 11.2 所示。

程序 11.2　计算中位数：

```
1:  def get_median(data):
2:      data = sorted(data)
3:      size = len(data)
4:      if size % 2 == 0:   #判断列表长度为偶数
5:          median = (data[size//2]+data[size//2-1])/2
6:          data[0] = median
7:      if size % 2 == 1:   #判断列表长度为奇数
8:          median = data[(size-1)//2]
9:          data[0] = median
10:     return data[0]
11: print(get_median([1,2,3,4,5]))
```

输出：

```
3
```

分析：

程序的第 2、3 行对数据列表进行排序，并获取排序后的列表长度；第 4 和 7 行使用 if 条件语句判断数据列表的长度的奇偶性后，进行中位数位置索引并返回中位数。

（3）众数：标志资料中出现次数最多的数值，不易受极值影响，但可靠性较差，如程序 11.3 所示。

程序 11.3　计算众数：

```
def get_mode(arr):
    mode = []
    arr_appear = dict((a, arr.count(a)) for a in arr);
    if max(arr_appear.values()) == 1:
        return
    else:
        for k, v in arr_appear.items():
            if v == max(arr_appear.values()):
                mode.append(k)
        return mode
Arr = [1, 2, 3, 4, 5, 6, 5]
print(get_mode(Arr))
```

输出：

```
[5]
```

（4）极差：标志资料中最大值与最小值之差，计算方便，但不能反映数据分布状况，如程序 11.4 所示。

程序 11.4　计算极差：

```
def get_range(arr):
    return max(arr)-min(arr)

List = [1, 2, 3, 4, 5, 6, 7]
print(get_range(List))
```

输出：

```
6
```

（5）方差：标志资料中各标志值与均值差值平方的均值，能够反映数据分布状况，但易受计量单位与均值影响，如程序 11.5 所示。

程序 11.5　计算方差：

```
def variance2(l):#平方-期望的平方的期望
    ex=float(sum(l))/len(l)
    s=0
    for i in l:
        s+=(i-ex)**2
    return float(s)/len(l)
List = [1, 2, 3, 4, 5, 6, 7]
print(variance2(List))
```

输出：

```
4.0
```

（6）标准差：即方差的平方根，与方差相似，如程序 11.6 所示。

程序 11.6　计算标准差：

```
import math
def variance3(l): #平方-期望的平方的期望
    ex=float(sum(l))/len(l)
        s=0
        for i in l:
            s+=(i-ex)**2
        return math.sqrt(float(s)/len(l))
```

```
List = [1, 2, 3, 4, 5, 6, 7]
print(variance3(List))
```

输出：

```
2.0
```

11.2　假 设 检 验

统计学方法包括统计描述和统计推断两种方法，其中，统计推断又包括参数估计和假设检验。

参数估计用样本统计量去估计总体的参数的真值，假设检验则是根据样本统计量来检验对总体参数的先验假设是否成立，其基本思想是：先提出假设 H_0，然后根据资料的特点（如检验统计量、抽样分布等），计算相应的统计量 μ，接着基于选取的显著性水平 α 确定原假设的接受域与拒绝域的临界值 $Z_{f(\alpha)}$，最后通过比较统计量 Z 与临界值 $Z_{f(\alpha)}$ 来判断假设是否成立。

资料总体均值检验的统计量计算的一般流程，如图 11.1 所示。

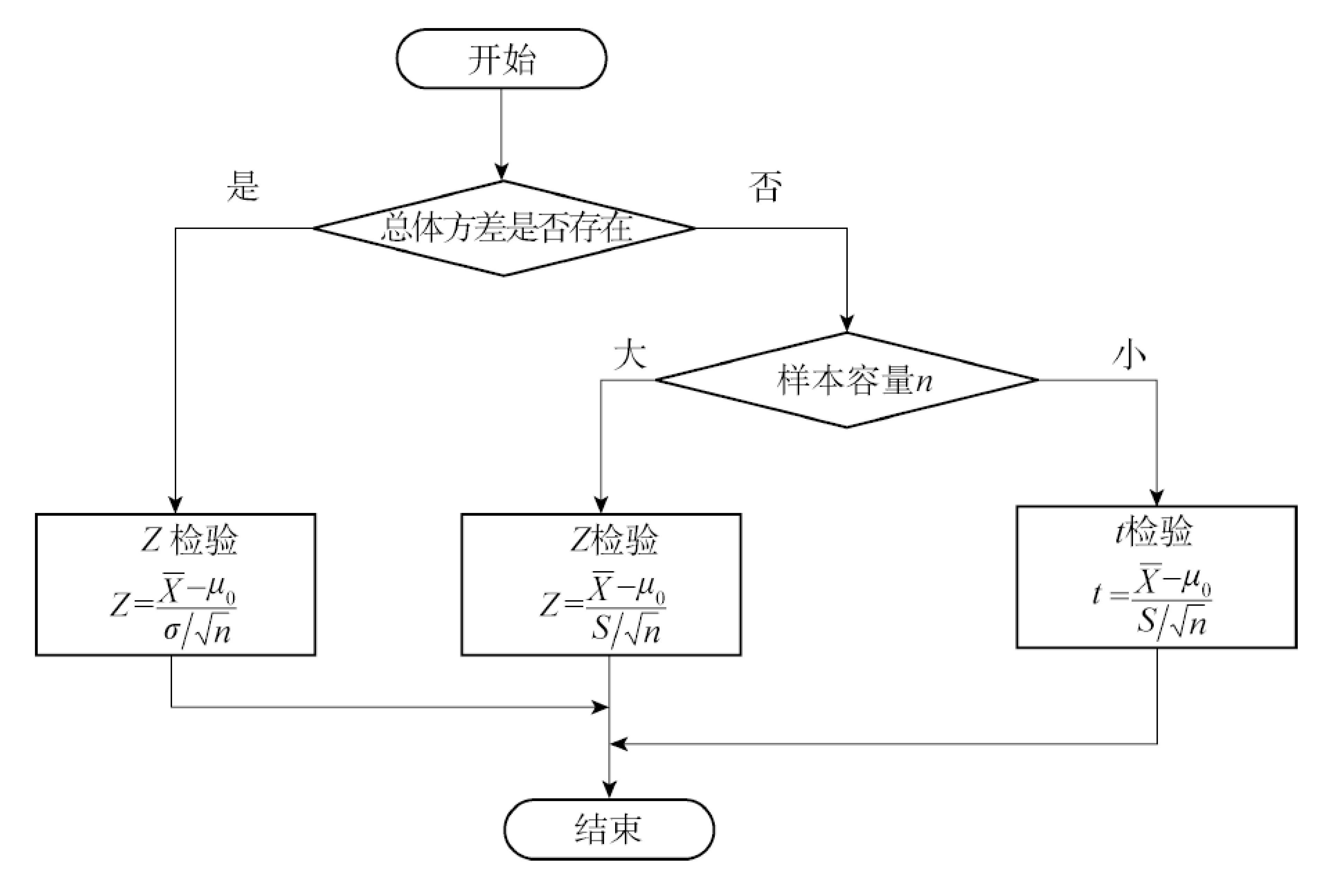


图 11.1　总体均值检验的统计量计算流程

我们将通过一个简单的实例来剖析其计算过程：设某皮鞋厂经理估计生产的皮鞋平均尺寸为 35 码，他从 2017 年皮鞋的生产记录中随机抽取 40 双，统计得到它们的尺寸数据如表 11.1 所示，请问根据该调查结果，经理的估计是否准确？

表 11.1　皮鞋尺寸抽样数据

29	31	36	34	37	27	34	39	29	35
22	34	20	37	39	27	29	36	39	22
30	33	29	27	35	37	26	32	28	33
38	36	39	42	38	36	21	19	26	29

对应于图 11.1 的计算流程，首先我们需要计算总体均值检验统计量，再通过比较该统计量与临界值，做出是否推翻原假设的决定，计算过程具体如程序 11.7 所示。

程序 11.7　皮鞋尺寸总体均值检验：

```
import numpy as np
def hypothesis_test(sample_set, var, z_list, t_list):
    u = 35
    test_type = "z_test"
    mean = np.mean(sample_set)
    num_sample = len(sample_set)
    if var is None:
        s = np.var(sample_set)
        if num_sample > 30:
            Z = (mean - u) / (s / np.sqrt(num_sample))
        else:
            test_type = "t_test"
            t = (mean - u) / (s / np.sqrt(num_sample))
    else:
        Z = (mean - u) / (var / np.sqrt(num_sample))
    a = 0.05
    if test_type == "z_test":
        z_thred = z_list.get(a)
        h0 = Z < z_thred[0] or Z > z_thred[1]
    else:
        t_thred = t_list.get(a)
        h0 = t< t_thred[0] or t > t_thred[1]
    print("The hypothesis 0 is:", h0)
```

输出：

```
The hypothesis 0 is: False
```

分析：

首先我们设皮鞋的平均尺寸 $H_0 : \mu = 35$，即总体皮鞋的平均尺寸与经理估计的 35 码没有差异，如程序的第 3 行所示。对照图 11.1，由于总体方差 var 未知，且样本数大于 30 为大样本，因此我们可以用样本标准差 s 代替总体标准差 var，计算得出统计量 $Z = -3.455$。设显著性水平 $\alpha = 0.05$，查表在 z_list 可得临界值 z_thred=(−1.96,1.96)，最后比较统计量 Z 与临界值 z_thred，因为 $Z = -3.455 < -1.96$，所以可以拒绝原假设 H_0，即我们认为经理的估计不准确。

假设检验在质量管理、过程控制、经济预测中有着非常广泛的应用，同时假设检验也有着很多优点，例如，有时候我们无法知道总体的具体参数，这时候我们就可以从总体中抽取样本来估计总体参数的范围。但是假设检验的缺点也是很明显的，它无法做到完全的精准，当资料本身没有可比性时也无法进行比较。所以在采用假设检验时，需要先分析自己的问题是否适合使用假设检验。

11.3　方 差 分 析

方差分析是通过划分误差来源来分析变量之间关系的一种方法。具体来看，它主要是用来分析分类型自变量与数值型因变量之间的关系。方差分析可以分为单因素方差分析、双因素方差分析和多因素方差分析。以下我们以单因素方差分析为例，学习方差分析的具体过程。

消费者协会为了对几个行业的服务质量进行评价，分别对四个行业抽取了不同企业在 2017 年接收到的投诉次数，如表 11.2 所示。

表 11.2　消费者对四个行业的投诉次数

No	Retail	Tourism	Airline	Industry
1	44	51	21	58
2	53	56	34	44

续表

No	Retail	Tourism	Airline	Industry
3	34	45	40	77
4	57	68	31	51
5	40	39	49	65
6	49	29		
7	66			

协会需要从中分析判断出这四个行业之间的服务质量是否有显著差异，即判断行业对投诉次数是否有显著影响。根据方差分析，上述判断最终被归结为检验这四个行业被投诉次数的均值是否相等，若均值相等，则说明行业对投诉次数没有影响，反之，则意味着它们间存在显著差异。在具体学习方差分析的计算步骤之前，我们先了解一些相关概念。

（1）因素（因子）：表示要检验的对象，协会要分析行业对投诉次数是否有影响，因此行业即为因子。

（2）水平（处理）：表示因子的不同表现，对应于上述例子，四个行业即为因子的水平。

（3）观察值：每个因子下的样本值，类似的，每个行业的被投诉次数即为观察值。

（4）总体：表示因子的每一个水平，上述的四个行业可分别看作一个总体。

（5）样本数据：被投诉次数即可看作四个总体的样本数据。

（6）随机误差：表示因子在同一总体下，样本各观察值之间的差异，例如，同一行业下不同企业被投诉的次数差异。

（7）系统误差：表示因子在不同总体下，例如，不同行业之间被投诉的次数差异，这种差异包括由于抽样导致的随机误差，与行业本身造成的系统误差。

（8）组内方差：表示因子在同一总体下样本数据的方差，例如，零售业被投诉次数的方差，值得注意的是组内方差只包含随机误差。

（9）组间方差：表示因子在不同总体下各样本之间的方差，如四个行业被投诉次数之间的方差，其中包括了随机误差与系统误差。

现在我们接着分析上述例子，首先假设四个行业被投诉的次数均值相等，即 $H_0: m_1 = m_2 = m_3 = m_4$，则备选假设 $H_1: m_i (i = 1,2,3,4)$ 不全相等。然后构造检验的统计量，具体计算步骤如下。

（1）计算总体均值。计算各行业样本观察值的公式为：

$$\bar{x}_i = \frac{\sum_{j=1}^{n_i} x_{i,j}}{n_i} \tag{11-1}$$

其中，$x_{i,j}$ 表示在第 i 个行业中的企业 j 的被投诉次数，n_i 表示行业 i 的样本数，由此各行业的均值为：

$$\bar{x}_{\text{retail}} = \frac{44+53+\cdots+49+66}{7} = 49$$

$$\bar{x}_{\text{tourism}} = \frac{51+56+\cdots+39+29}{6} = 48$$

$$\bar{x}_{\text{airline}} = \frac{21+34+\cdots+31+49}{5} = 35$$

$$\bar{x}_{\text{industry}} = \frac{58+44+\cdots+51+65}{5} = 59$$

（2）计算全部观察值的均值。计算全部观察值均值的公式为：

$$\bar{x} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n_i} x_{i,j}}{n} = \frac{\sum_{i=1}^{k} n_i \bar{x}_i}{n}, k=4 \tag{11-2}$$

其中，$x_{i,j}$ 表示在第 i 个行业中的企业 j 的被投诉次数，k 表示水平数，n_i 表示行业 i 的样本数，n 表示全部样本数，由此全部观察值的均值为：

$$\bar{x} = \frac{7\cdot 49+6\cdot 48+5\cdot 35+5\cdot 59}{7+6+5+5} = 47.87$$

（3）计算方差。计算组内方差（MSE）与组间方差（MSA），其公式分别为：

$$MSE = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n_i} (x_{i,j} - \bar{x}_i)^2}{n-k} \tag{11-3}$$

其中，$x_{i,j}$ 表示在第 i 个行业中的企业 j 的被投诉次数，$\bar{x}_i$ 表示行业 i 的均值，n 表示全部样本数，k 表示水平数。

$$MSA = \frac{\sum_{i=1}^{k} n_i (\bar{x}_i - \bar{x})^2}{k-1} \tag{11-4}$$

其中，$\overline{x}_i$ 表示行业 i 的均值，$\overline{x}$ 表示全部观察值的均值，k 表示水平数。对应案例中的组内方差与组间方差分别为：

$$MSE=\frac{(44-49)^2+\cdots+(65-59)^2}{23-4}=142.53$$

$$MSA=\frac{7\cdot(49-47.87)+\cdots+5\cdot(59-47.87)}{4-1}=485.54$$

（4）计算检验统计量 F 。对比组内方差与组间方差，计算检验统计量 F ：

$$F=\frac{MSA}{MSE}=\frac{485.54}{142.53}=3.41$$

将统计量 F 的值与给定显著性水平 α 的临界值 F_α 进行比较，若 $F>F_\alpha$，则拒绝原假设 H_0，否则将不拒绝原假设 H_0 。我们设 $\alpha=0.005$ ，即 $F_\alpha=5.73$，因为 $F=3.41<5.73$ ，所以我们不能认为所检验的因素对观察值有显著影响。

11.4　统计回归分析

在第 10 章中，我们曾经提到过“回归”，“回归”一词源于 19 世纪生物学家 Galton 的遗传学研究：子辈的身高有回到父辈平均身高的趋势。他通过观察 1078 对夫妇与子女，以每对夫妇的平均身高作为 x，取他们成年子女的身高为 y，将结果在二维平面上绘成散点图，发现趋势近乎一条直线，其直线方程为 $y=33.73+0.516x$ ，该趋势及回归方程表明父母平均身高 x 每增加一个单位，其成年子女的身高 y 也平均增加 0.516 个单位。这种利用数据统计原理，确定大量统计数据中因变量与某些自变量的相关关系，并用于预测今后的因变量变化的方法就称为回归分析。

11.4.1　线性回归模型

依旧使用上述例子，如果现在给定类似的样本数据，如表 11.3 所示，我们将如何构建并计算父辈与子辈身高的回归方程呢？

表 11.3　父辈与子辈身高统计表

父辈平均身高（m）	子辈身高（m）
1.62	1.7
1.65	1.72
1.7	1.73
1.74	1.78
1.78	1.8
1.83	1.86

1. 构建回归方程

首先我们需要把自变量都提取出来，构成特征变量 X，该变量是一个向量，即 $X=[x_1,x_2,\cdots,x_m]$。我们可以假设一个线性函数：

$$h_\theta(X)=\theta_0+\theta_1x_1+\theta_2x_2+\cdots++\theta_mx_m \tag{11-5}$$

其中，θ 为特征参数，该参数的大小决定了特征变量 x_i 对估计的影响程度。这里我们提取的特征变量仅为父辈身高，因此实际的回归模型为：

$$h_\theta(X)=\theta_0+\theta_1x_1 \tag{11-6}$$

2. 建立损失模型

为了达到最好的拟合效果，必须有一个函数去衡量模型的误差。一般把这个函数定义为损失函数 J：

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}-h_\theta(x^{(i)}))^2 \tag{11-7}$$

其中，$h_\theta(x^{(i)})$ 表示线性函数 $h_\theta(X)$ 关于样本 $x^{(i)}$ 估计值，$y^{(i)}$ 则表示样本 $x^{(i)}$ 的真实值。这个损失函数是对估计值与真实值的差的平方进行估计。我们的目标是使拟合的损失最小，即最小化 J 函数的输出，这样求得的 θ 值便是我们理想中的特征参数。下面通过介绍梯度下降法，来降低拟合损失，我们将学习到如何使用该方法求得线性函数的最佳特征参数。

3. 训练模型

以爬山为例，我们在一座大山上的某处位置，由于不知道怎么下山，于是决定走一步算一步，也就是在每走到一个位置的时候，求解当前位置的梯度，沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，向这一步所在位置沿着

最陡峭最易下山的位置走一步。一直走到山脚。当然有可能我们不能走到山脚，而是到了某一个局部的山峰低处。在具体了解梯度下降的算法之前，先看看一些相关概念。

梯度。在微积分中，梯度以向量的形式表示多元函数对参数的 ∂ 偏导，例如，函数 $g(x,y)$ 分别对 x,y 求偏导，梯度向量即为 $(\partial g/\partial x,\partial g/\partial y)^{\mathrm{T}}$，简称 grad $g(x,y)$ 或者 $\nabla g(x,y)$。从几何意义上讲，梯度就是函数变化最快的方向。具体来说，对于函数 $g(x,y)$ 在点 (x_0,y_0)，梯度向量 $(\partial g/\partial x_0,\partial g/\partial y_0)^{\mathrm{T}}$ 就是 $g(x,y)$ 变化最快的方向，即沿着梯度向量的方向，更加容易找到函数的极值。

步长（学习率）。步长决定了在梯度下降迭代过程中，每一步沿梯度方向前进的长度，即爬山例子中，所在位置沿最陡峭方向下山走的那一步长度。

（1）计算损失函数梯度。针对上文的损失函数 J，对应的图像如图 11.2 所示。

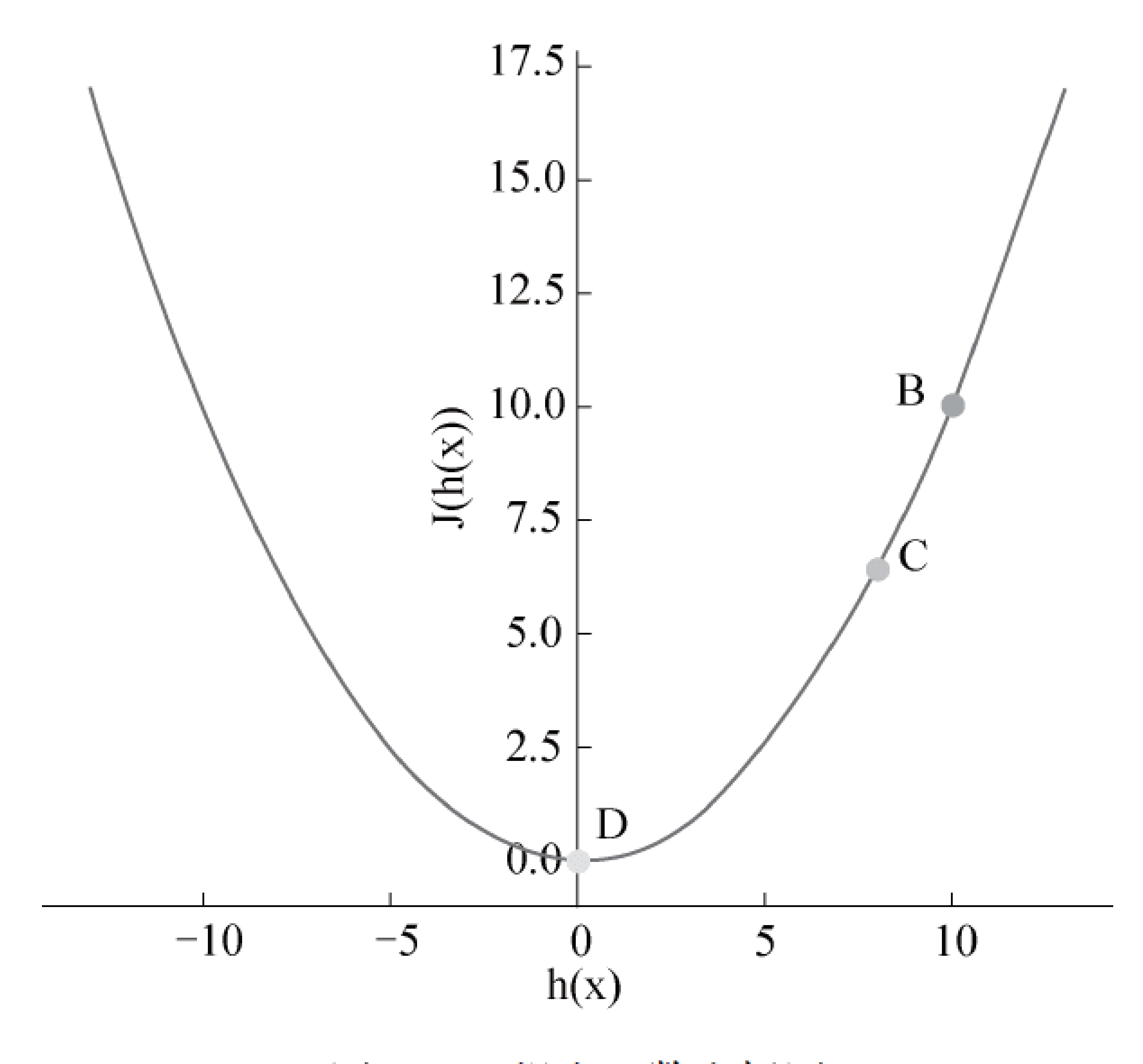


图 11.2　损失函数坐标图

假设线性函数 $f_\theta(x)$ 在图中的 B 点，求解梯度首先需要对 J 在 B 点的 θ_i 参数求导，其表达式如（11-8）所示。

$$\text{grad } J(\theta_j)=\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1,\cdots,\theta_n)=\frac{2}{m}\sum_{i=1}^{m}(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)} \tag{11-8}$$

（2）更新特征参数。在得到函数 J 在 B 点的梯度（移动方向）后，定义步长 α，即每

一步沿梯度方向前进的长度，并将其代入特征参数的更新公式（11-9）中：

$$\theta_i = \theta_i + \alpha \cdot \text{grad } J(\theta_i) \tag{11-9}$$

在更新完特征参数，$f_\theta(x)$ 从点 B 降到点 C 后，计算 $f_\theta(x)$ 在 C 点的损失值，若该值小于一定阈值或达到最大迭代次数，则返回特征参数，否则将继续更新特征参数，直到损失函数收敛或达到迭代限制条件。

11.4.2　实例说明

在理解了上述的步骤后，我们创建名为 linear_regression.py 文件，并在 linear_regression.py 文件中创建 Linear_regression 类，如程序 11.8 所示。

程序 11.8　创建 Linear_regression 类：

```
class Linear_regression(object):
    def __init__(self, weights, learning_rate, num_iter, threshold ):
        """
        输入：线性回归实例 self，特征参数 weights，步长 learning_rate，
        最大迭代次数 num_iter，回归模型误差阈值 threshold
        输出：无
        描述：线性回归构造函数
        """
        self.weights = weights
        self.learning_rate = learning_rate
        self.num_iter = num_iter
        self.threshold = threshold
        super(Linear_regression, self).__init__()
```

首先我们以表 11.3 的数据为例，假设预测函数 $h_\theta(x)$ 是一个一元一次方程：

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

初始化其特征参数 $\theta_0 = 0.45$，$\theta_1 = 1$，然后根据损失函数 J 计算梯度 $\text{grad } J(\theta_i)$：

$$\text{grad } J(\theta_0) = \frac{2}{m}\sum_{i=1}^{m}(y^{(i)} - \theta_1 x_1^{(i)} - \theta_0)$$

$$\text{grad } J(\theta_1) = \frac{2}{m}\sum_{i=1}^{m}x_1^{(i)}(y^{(i)} - \theta_1 x_1^{(i)} - \theta_0)$$

将表 11.3 的数据依次代入公式，得到 $\text{grad } J(\theta_0) = -2.43$，$\text{grad } J(\theta_1) = -4.1878$，梯度计算结束。该过程对应到 Linear_regression 类的方法如程序 11.9 所示。

程序 11.9　计算损失函数梯度：

```
def compute_grandient(self, dataset, labels):
    """
    输入：线性回归实例 self，训练数据集 dataset，标签集合 labels
    输出：损失函数梯度 w_gradient
    描述：计算损失函数梯度
    """
    m = float(len(dataset))
    h = dataset * self.weights
    w_gradient = (2 / m) * dataset.transpose() * (labels - h)
    return w_gradient
```

然后我们通过定义的步长 $\alpha = 0.01$，更新特征参数：

$$\theta_0 = 0.45 - 0.01 \times 2.43 = 0.4257$$

$$\theta_1 = 1 - 0.01 \cdot 4.1878 = 0.9581$$

之后将更新后的参数代入损失函数，计算损失值 $J = 0.09564$，在其不满足结束条件的情况下，继续更新特征参数，直到损失小于阈值或达到最大迭代限制。特征参数更新对应到 Linear_regression 类的方法如程序 11.10 所示。

程序 11.10　更新特征参数：

```
def update_weights(self, w_gradient):
    """
    输入：线性回归实例 self，损失函数梯度 w_gradient
    输出：无
    描述：根据损失函数梯度，更新特征参数
    """
    self.weights += self.learning_rate * w_gradient
```

现在我们尝试着将前面的知识点串联起来，从而梳理出一个完整的线性回归算法流程，如图 11.3 所示。

参照完整的线性回归算法流程图，我们在 Linear_regression 类中编写程序如程序 11.11 所示。

程序 11.11　线性回归训练方法：

```
def fit(self, dataset, labels):
    """
    输入：线性回归实例 self，训练数据集 dataset，标签集合 labels
    输出：无
    描述：利用梯度下降法，迭代训练模型特征参数
```

```
        """
        _, n = np.shape(dataset)
        self.weights = np.ones((n, 1))
        for i in range(self.num_iter):
            self.gradient_descend(dataset, labels)
            h = dataset * self.weights
            error = np.math.fabs(np.sum((labels - h), axis=0)[0, 0])
            if error <= self.threshold:
                break

    def gradient_descend(self, dataset, labels):
        """
        输入：线性回归实例 self，训练数据集 dataset，标签集合 labels
        输出：无
        描述：利用损失函数梯度更新特征参数
        """
        w_gradient = self.compute_grandient(dataset, labels)
        self.update_weights(w_gradient)
```

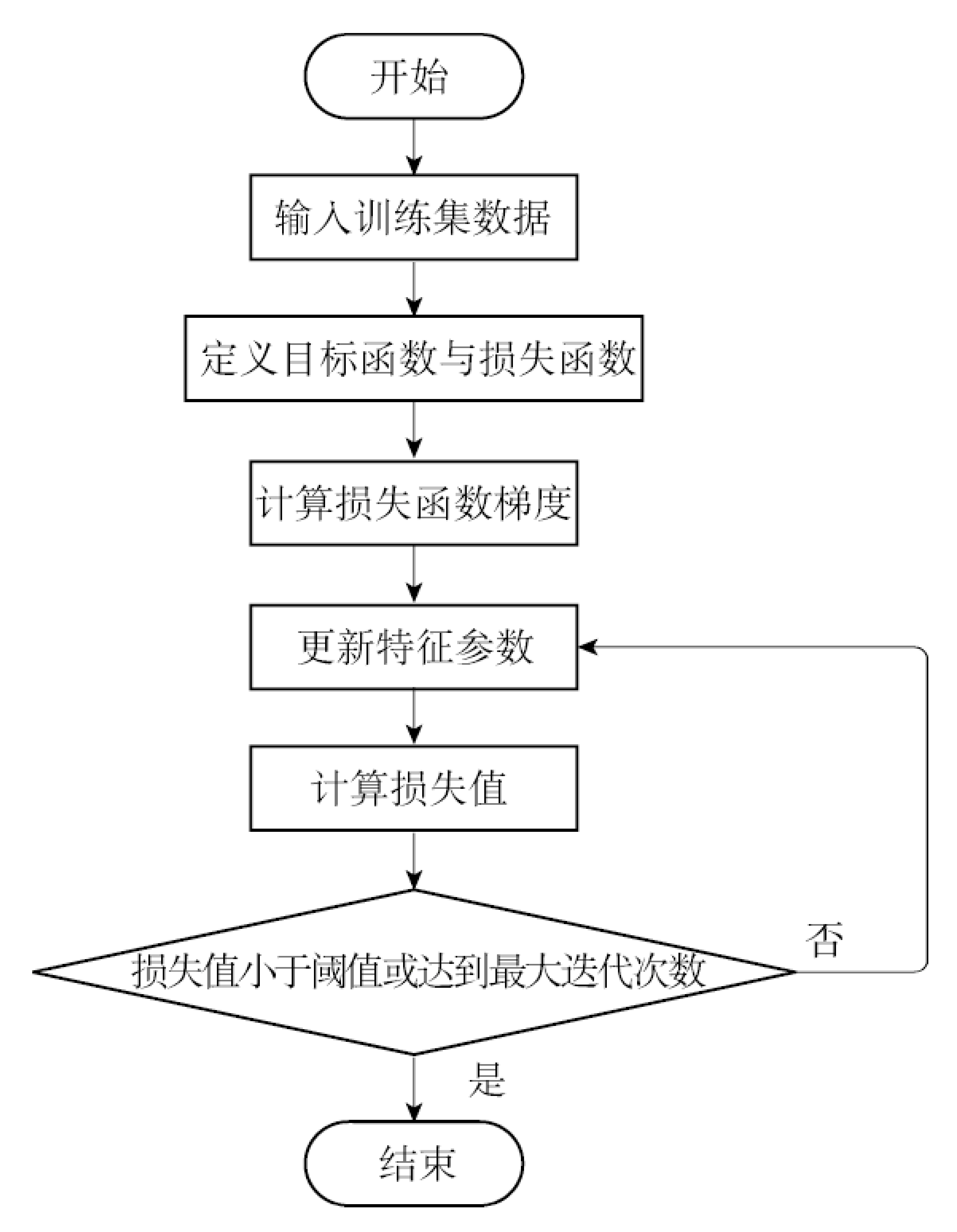


图 11.3　线性回归算法流程图

阅读至此，我们已经完成了对线性回归算法训练阶段的学习，接下来将是本文简单但有实际意义的一环：线性回归预测，对于一个测试对象，我们只需依次输入该对象的特征值，Linear_regression 类将根据以前的训练模型自动预测该对象的值，其具体程序如程序 11.12 所示。

程序 11.12　线性回归预测方法：

```
def predict(self, testset):
    """
    输入：线性回归实例 self，测试数据集 testset
    输出：预测值
    描述：利用线性回归训练模型，预测测试数据值
    """
    estimate = testset * self.weights
    return estimate
```

最后我们只需通过一个数据加载和入口函数就可得到一个可运行的代码实例，其代码如程序 11.13 所示。

程序 11.13　数据加载函数与入口函数：

```
import numpy as np

def load_dataset(filename, delimiter=','):
    """
    输入：数据文件名 filename，数据分隔符 delimiter
    输出：训练数据集合，标签集合
    描述：加载训练数据集合，标签集合
    """
    data = np.loadtxt(filename, delimiter=delimiter)
    dataset = np.concatenate((np.ones((len(data), 1)), data[:, :-1]), axis=1)
    labels = data[:, -1]
    return np.mat(dataset), np.mat(labels).transpose()

if __name__ == '__main__':
    filename = 'height.data'
    dataset, labels = load_dataset(filename)
    linear_regression = Linear_regression()
    linear_regression.fit(dataset, labels)
    testset = np.mat([1,1.65])
    print(linear_regression.predict(testset))
```

11.5　总　结

我们在本章中回顾了统计学中计量资料的统计描述，有均值、中位数、极差、方差等，并具体学习了假设检验与线性回归分析模型与计算流程，如在假设检验中，提出假设、计算检验统计量、设置显著性水平、根据统计量与临界值决定是否推翻假设；线性回归算法中，线性函数与损失函数的构造，基于梯度下降法的特征参数迭代更新等。

11.6　练　习

（1）定义一个函数 statistic，要求在函数内随机生成 1000 个自然数的列表，并分别统计列表中数值的算术均值、中位数、总体标准差等。

（2）根据上述消费者协会的案例，编写函数 analyze_variance(complaint_dict)，实现方差分析具体逻辑。

（3）某省 1978～1986 年居民消费品购买力和居民货币收入统计如表 11.4 所示。

表 11.4　居民消费品购买力和居民货币收入统计

年份	居民消费品购买力 x	居民货币收入 y
1978	8.5	11.6
1979	11.1	14.1
1980	13.6	17.1
1981	15.8	19.6
1982	17.6	22.1
1983	20.5	25.6
1984	27.8	33.6
1985	33.5	40.5
1986	39.2	47.8

请求出该省居民消费品购买力和居民货币收入之间的回归方程。

第 12 章　数据管理与分析

12.1　基于 Python 的数据管理与分析

数据分析是基于某种行业目的，有导向地进行收集、整理、加工和分析数据的过程，在此过程中，我们可以使用统计分析、数据挖掘等方式或方法提取有用的信息，并进行概括与总结。如经典的“啤酒与尿布”营销案例：20 世纪 90 年代的美国沃尔玛超市中，“啤酒”与“尿布”两件商品会经常出现在年轻的父亲的购物篮中，超市的管理人员跟进研究发现，在美国有婴儿的家庭中，一般由母亲在家照看婴儿，年轻的父亲出门购买尿布。父亲在购买尿布的同时，往往会顺便为自己购买啤酒，这样就出现了啤酒与尿布这样两件看上去毫无关联的商品经常会出现在同一个购物篮的现象。

在得此结论后，沃尔玛开始尝试着将啤酒与尿布摆放在相同的区域，让年轻的父亲可以更便捷地找到这两件商品，并为获取更好的商品销售收入，进而推出一次可同时购买“啤酒与尿布”套餐的活动。由此可见，数据分析挖掘与概括的潜在信息和事物规律，对于我们日常的生活、工作等方面具有一定的指导意义，并且随着互联网的蓬勃发展，越来越多的应用涉及大数据，在这些大数据的背后潜在的又是怎样的数据规律，都需要我们进一步的研究与探索，基于如此的认识，数据分析常用的理论与方法又有哪些呢？

（1）统计分析：统计是进行科学研究的重要方法，通过数字揭示事物在特定时间方面的数量特征，以便对事物进行定量乃至定性分析，从而做出正确的决策，其常见的方法有描述性统计分析、集中趋势分析、相关分析、方差分析、回归分析等。

（2）数据挖掘：数据挖掘是数据分析的理论核心，各种数据挖掘的算法基于不同的数据类型和数据关联方式，迭代训练关系模型，深入数据内部，挖掘数据潜在价值，其中经典的算法有 C4.5 决策树、K-means 聚类、SVM 分类、Apriori 算法等。

为了更方便地统计与挖掘数据，许多优秀的统计分析的语言和工具被我们开发与使用，就如 Python 语言。Python 是一种面向对象的解释型计算机程序设计语言，它具有丰富而强大的库，如专用的科学计算扩展模块 numpy、scipy 和 matplotlib，以及机器学习、数据挖

掘模块 scikit-learn、tensorflow 等，它们为 Python 提供了快速数组处理、数值运算、模型训练以及绘图等功能，为数据分析提供了简洁、方便的调用接口。下面我们将以统计学生成绩为例，具体介绍基于 Python 语言的数据分析过程。

12.2　数据的导入与导出

数据分析的第一阶段是获取数据，通常情况下，数据会以文本文件或数据库链接等形式呈现给我们，我们要做的工作则是通过一些加载模块完成数据从文件到程序，或从程序到文件的转换操作。这是一项艰巨且重要的工作，但 Python 将使该工作变得异常简单，只需几行代码，就可轻松导入和导出数据了，下面我们将分别介绍以下几种存储文件基于 Python 的导入、导出操作：

（1）CSV

（2）Excel

（3）SQL

12.2.1　CSV 文件的导入与导出

1. CSV 文件的导入

我们通过人工统计，手动生成的学生成绩数据文件，如表 12.1 所示。

表 12.1　学生成绩表

Student ID	Name	Hour	Chinese	Math	English
1807022101	Alan	6	80	80	80
1807022102	Bing	10	78	90	94
1807022103	Ben	8	67	87	90
1807022104	Calvin	9	76	78	98
1807022105	Carter	8	87	78	79

接下来的工作是把该数据文件导入到程序中，其具体导入过程参见程序 12.1。

程序 12.1　CSV 文件的导入：

```
import pandas as pd
file_path = " report.csv"
df = pd.read_csv(file_path, encoding='utf-8')
print(df.head())
```

输出：

	Student ID	Name	Hour	Chinese	Math	English
0	1807022101	Alan	6	80	80	80
1	1807022102	Bing	10	78	90	94
2	1807022103	Ben	8	67	87	90
3	1807022104	Calvin	9	76	78	98
4	1807022105	Carter	8	87	78	79

分析：

在导入文件程序的开头，需要先引入 Python 的 pandas 模块，pandas 库提供了大量能使我们快速便捷地处理数据的函数和方法，在确定了 CSV 文件的所在路径后，调用 pandas 的 read_csv 方法，传入文件路径并指定编码方式，这样就完成了文件数据的导入工作，最后得到一个 DataFrame 对象 df，我们可以通过访问该对象的属性，获取特定单元的值。

2. CSV 文件的导出

假设我们已经完成成绩分析，并得到了相关统计信息，之后要做的就是把这些分析数据保存为文件，存储到磁盘中，其具体过程参见程序 12.2。

程序 12.2　CSV 文件的导出：

```
import pandas as pd
subject = ['Chinese', 'Math', 'English']
value = [77.6, 82.5, 88.2]
mean_list = list(zip(subject, value))
df = pd.DataFrame(data=mean_list, columns=['Subject', 'Means'])
df.to_csv('subject_means.csv', index=False, header=True)
```

输出：

```
Subject,Means
Chinese,77.6
Math,82.5
English,88.2
```

分析：

同样，在导出数据到文件的过程中，仍是借助于 pandas 模块，在我们模仿一些分析数据后，调用 pandas 的 DataFrame 类的构造函数，传入数据与列名，在得到 DataFrame 对象后，再通过 to_csv 方法，指定导出路径与其他一些选项，如是否生成索引、表头等，完成数据的导出工作。

12.2.2　Excel 文件的导入与导出

1. Excel 文件的导入

同样以表 12.1 中的数据为例，Python 将 Excel 格式数据导入的具体过程参见程序 12.3。

程序 12.3　Excel 文件的导入：

```
import pandas as pd
file_path = "report.xlsx"
df = pd.read_excel(file_path)
print("the table of report:\n{0}\n".format(df))
```

输出：

the table of report:

	Student ID	Name	Hour	Chinese	Math	English
0	1807022101	Alan	6	80	80	80
1	1807022102	Bing	10	78	90	94
2	1807022103	Ben	8	67	87	90
3	1807022104	Calvin	9	76	78	98
4	1807022105	Carter	8	87	78	79

分析：

与导入 CSV 文件类似，需要先引入 pandas 模块，确定 Excel 文件路径，再调用 pandas 的 read_excel 方法，传入文件路径，得到一个 DataFrame 对象 df，之后就可以通过操作该对象的属性，访问或修改特定单元的值。

2. Excel 文件的导出

我们在得到统计信息后，如果需要将该信息以 Excel 格式存储到磁盘，那么同样将使用 pandas 模块，其具体过程参见程序 12.4。

程序 12.4　Excel 文件的导出：

```
import pandas as pd
subject = ['Chinese', 'Math', 'English']
value = [77.6, 82.5, 88.2]
mean_list = list(zip(subject, value))
df = pd.DataFrame(data=mean_list, columns=['Subject', 'Means'])
writer = pd.ExcelWriter('subject_means.xlsx', engine='xlsxwriter')
df.to_excel(writer, sheet_name='Sheet1')
writer.save()
```

输出：

```
Subject,Means
Chinese,77.6
Math,82.5
English,88.2
```

分析：

与导出 CSV 文件不同的是，我们在依次导入 pandas 模块、模拟统计数据、生成 DataFrame 对象后，调用 ExcelWriter 方法，传入导出路径与 xlsx 解析引擎，得到 Excel 文件的输出流，接着将 df 对象绑定到输出流，并将输出流通过 save 方法，生成 Excel 文件到指定路径。

12.2.3　SQL 数据的导入与导出

1. SQL 数据的导入

除了上述文件外，数据也可能来源于数据库，我们以导入 SQLite 数据库数据为例，具体程序如程序 12.5 所示。

程序 12.5　SQLite 数据的导入：

```
import pandas as pd
from sqlalchemy import create_engine
db_file = r'report.db'
engine = create_engine(r"sqlite:///{}".format(db_file))
sql = 'select * from tb_report'
df = pd.read_sql(sql, engine)
print("the table of report:\n{0}\n".format(df))
```

输出：

```
the table of report:
     Student ID    Name     Hour   Chinese   Math   English
0    1807022101    Alan     6      80        80     80
1    1807022102    Bing     10     78        90     94
2    1807022103    Ben      8      67        87     90
3    1807022104    Calvin   9      76        78     98
4    1807022105    Carter   8      87        78     79
```

分析：

可以看到，我们在加载 SQLite 数据前，依次导入了 pandas 模块和 sqlalchemy 的 create_engine 函数，之后开始创建数据库引擎，并将其关联数据库文件 db_file，最后通过 sql 的查询语句检索出学生的成绩信息。

2. SQL 数据的导出

与上述导出方式不同，接下来我们将统计数据保存到 SQLite 数据库中，如程序 12.6 所示。

程序 12.6　SQLite 数据的导出：

```
import pandas as pd
import sqlite3 as sql
subject = ['Chinese', 'Math', 'English']
value = [77.6, 82.5, 88.2]
mean_list = list(zip(subject, value))
df = pd.DataFrame(data=mean_list, columns=['Subject', 'Means'])
db_file = r'report_stat.db'
conn = sql.connect(db_file)
df.to_sql('tb_stat',conn,flavor='sqlite',schema=None,if_exists='replace',index=False,
          index_label=None,chunksize=None,dtype=None)
conn.close()
```

分析：

我们在把数据导出到 SQLite 数据库时，除了引入 pandas 模块，还引入了 sqlite3 的 sql 模块，并将统计数据封装为 DataFrame 对象，接着调用 sql 模块的 connect 方法，连接数据库，之后通过 DataFrame 对象的 to_sql 方法将数据导出到数据库文件 report_stat.db 的 tb_stat 表中，最后关闭数据库连接。

12.3　数 据 分 析

接下来将进入数据分析的第二阶段——数据分析。在本阶段，我们通过使用 Python 的库函数轻松完成信息的计量统计、数据的关联度分析等，仅仅是一小段代码，我们就可以实现：

（1）计算资料描述统计量。

（2）计算数据关联度。

（3）数据间的线性回归分析。

12.3.1　聚合统计量分析

同样以表 12.1 中的数据为例，我们可以通过统计各学科成绩的算术平均数、中位数、极差、标准差等，以了解其分布或是趋势等，具体程序如程序 12.7 所示。

程序 12.7　计算资料描述统计量：

```
import pandas as pd
file_path = "report.csv"
df = pd.read_csv(file_path, encoding='utf-8')
print("the means of Chinese:{0}".format(df['Chinese'].mean()))
print("the median of Chinese:{0}".format(df['Chinese'].median()))
print("the span of Chinese:{0}".format(df['Chinese'].max() - df['Chinese'].min()))
print("the standard of Chinese:{0}".format(df['Chinese'].std()))
print("\n")
print("the means of Math:{0}".format(df['Math'].mean()))
print("the median of Math:{0}".format(df['Math'].median()))
print("the span of Math:{0}".format(df['Math'].max() - df['Math'].min()))
print("the standard of Math:{0}".format(df['Math'].std()))
print("\n")
print("the means of English:{0}".format(df['English'].mean()))
print("the median of English:{0}".format(df['English'].median()))
print("the span of English:{0}".format(df['English'].max() - df['English'].min()))
print("the standard of English:{0}".format(df['English'].std()))
```

输出：

```
the means of Chinese:77.6
the median of Chinese:78.0
```

```
the span of Chinese:20
the standard of Chinese:7.231873892705818

the means of Math:82.6
the median of Math:80.0
the span of Math:12
the standard of Math:5.5497747702046425

the means of English:88.2
the median of English:90.0
the span of English:19
the standard of English:8.438009243891594
```

分析：

在上述程序中，我们通过调用 pandas 模块的函数，分别统计出学生各科成绩的平均值、中位数、极差与标准差等，稍加分析可以看出，学生的英语平均成绩较为突出，但极差与标准差统计值较大，说明学生整体英语水平波动较大，而在数学成绩中，其平均成绩与波动状况等都表现稳定，但学生的语文成绩则完全与之相反，有待进一步巩固加强。

12.3.2　关联度分析

关联度分析是指对两个或多个具备相关性的变量元素进行分析，从而衡量两个变量因素的相关密切程度。在本节中，我们将通过协方差、相关系数与线性回归分析等方面挖掘数据间的关联关系。

1. 协方差分析

协方差一般用来衡量两个变量的总体误差，如果两个变量的变化趋势一致，协方差就是正值，说明两个变量正相关。如果两个变量的变化趋势相反，协方差就是负值，说明两个变量负相关。如果两个变量相互独立，那么协方差就是 0，说明两个变量不相关。以下是协方差的计算公式：

$$\mathrm{cov}(X,Y)=\frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{n-1} \tag{12-1}$$

在表 12.1 中，除了学生成绩，还可以看到另一个关键信息——学习时长，现在我们基

于协方差来分析学生每天的学习用时与本次综合成绩的关联关系，具体程序如程序 12.8 所示。

程序 12.8　学习时长与综合成绩的协方差：

```
import numpy as np
import pandas as pd
file_path = "report.csv"
df = pd.read_csv(file_path, encoding='utf-8')
print("the table of report:\n{0}".format(df))
report = list(df.iloc[:, 3:].sum(axis=1))
hour_report = list(df["Hour"])
hour_report.extend(report)
data = np.array(hour_report).reshape(5, 2, order="F")
df_hour_report = pd.DataFrame(data, columns=['Hour', 'Report'])
print("the table of hour_report:\n{0}".format(df_hour_report))
print("\n")
print("the covariance of hour_report:\n{0}".format(df_hour_report.cov()))
```

输出：

```
the table of report:
   Student ID    Name     Hour   Chinese   Math   English
0  1807022101    Alan     6      80        80     80
1  1807022102    Bing     10     78        90     94
2  1807022103    Ben      8      67        87     90
3  1807022104    Calvin   9      76        78     98
4  1807022105    Carter   8      87        78     79

the table of hour_report:
   Hour  Report
0     6     240
1    10     262
2     8     244
3     9     252
4     8     244

the covariance of hour_report:
         Hour    Report
Hour      2.2     11.9
Report   11.9     76.8
```

分析：

在成功导入数据后，首先抽取了学生的各门学科成绩计算累加和，并将其和学生每天的学习时长合并，创建出一个新的 DataFrame 对象 df_hour_report，接着调用 df_hour_report 的 cov 方法，计算学习时长与综合成绩的协方差 $\text{cov}(hour, report) = 11.9$，说明有效学习时长与综合成绩正相关，即学生每天有效的学习用时越长，综合成绩越高。

2. 相关系数分析

协方差可以通过数字衡量变量间的相关性，却无法针对相关的密切程度进行度量，如学生每日有效的学习用时与综合成绩的相关程度，当我们面对多个变量时，无法通过协方差来说明哪两组数据的相关性最高。要衡量和对比相关性的密切程度，就需要使用下一个方法——相关系数。

相关系数是反应变量之间关系密切程度的统计量，相关系数的取值区间在−1～1 之间。1 表示两个变量完全线性相关，−1 表示两个变量完全负相关，0 表示两个变量不相关。数据越趋近于 0 表示相关关系越弱，其计算公式如（12-2）所示。

$$r_{xy} = \frac{S_{xy}}{S_x S_y} \tag{12-2}$$

其中，r_{xy} 表示样本相关系数，S_{xy} 表示样本协方差，S_x 表示 x 的样本标准差，S_y 表示 y 的样本标准差。接下来我们将相关系数代入学生成绩的例子，具体程序如程序 12.9 所示。

程序 12.9　学习时长与综合成绩的相关系数：

```
import math
import numpy as np
import pandas as pd
file_path = "report.csv"
df = pd.read_csv(file_path, encoding='utf-8')
print("the table of report:\n{0}\n".format(df))
report = list(df.iloc[:, 3:].sum(axis=1))
hour_report = list(df["Hour"])
hour_report.extend(report)
data = np.array(hour_report).reshape(5, 2, order="F")
df_hour_report = pd.DataFrame(data, columns=['Hour', 'Report'])
print("the covariance of hour_report:\n{0}\n".format(df_hour_report.cov()))
matrix_cov = df_hour_report.cov()
coefficent_cor = round(
```

```
            matrix_cov["Hour"]["Report"] / math.sqrt(matrix_cov["Hour"]["Hour"]) /
                math.sqrt(matrix_cov["Report"]["Report"]), 2)
print("the coefficent between hour and report is:{0}".format(coefficent_cor))
```

输出：

```
the table of report:
   Student ID     Name     Hour     Chinese    Math     English
0  1807022101     Alan     6        80         80       80
1  1807022102     Bing     10       78         90       94
2  1807022103     Ben      8        67         87       90
3  1807022104     Calvin   9        76         78       98
4  1807022105     Carter   8        87         78       79

the covariance of hour_report:
         Hour   Report
Hour      2.2    11.9
Report   11.9    76.8

the coefficent between hour and report is:0.92
```

分析：

我们基于协方差的计算流程，在得到时长与成绩的协方差后，将相关变量代入公式（12-2），最后得出有效学习时长与综合成绩的相关系数 $correlation_coefficent = 0.92$，从数值的层面可以说明，学生每天有效的学习时长与综合成绩正相关，且两者的关联性强。

3. 线性回归分析

我们在得出时长与成绩具有较强关联的结论后，可尝试着使用回归分析描述出其对应的关联关系，回归分析是确定两组或两组以上变量间关系的统计方法。回归分析按照变量的数量分为一元回归和多元回归。我们的数据中只包含学习时长和综合成绩两个变量，因此使用一元回归，接下来将通过调用 scikit-learn 模块完成两者线性模型的学习，具体程序如程序 12.10 所示。

程序 12.10　学习时长与综合成绩的线性模型：

```
import pandas as pd
from sklearn.linear_model import LinearRegression
file_path = "report.csv"
df = pd.read_csv(file_path, encoding='utf-8')
```

```
print("the table of report:\n{0}\n".format(df))
hour = df["Hour"].values.reshape(5, 1)
report = df.iloc[:, 3:].sum(axis=1).values.reshape(5, 1)
linear_reg = LinearRegression()
linear_reg.fit(hour, report)
print("the correlation formula between hour and report is:\n" +
        "y={0}x+{1}".format(linear_reg.coef_[0][0], linear_reg.intercept_[0]))
```

输出：

```
the table of report:
   Student ID    Name    Hour   Chinese   Math   English
0  1807022101    Alan    6      80        80     80
1  1807022102    Bing    10     78        90     94
2  1807022103    Ben     8      67        87     90
3  1807022104    Calvin  9      76        78     98
4  1807022105    Carter  8      87        78     79

the correlation formula between hour and report is:
y=5.40909091x+204.04545455
```

分析：

在案例中，综合成绩是随着有效学习时长的变化而改变的，因此我们在导入数据到程序后，首先抽取出学生学习时长与各科成绩数据，将学习时长设置为自变量 x，并统计出综合成绩后，将其设置为因变量 y，之后分别将变量作为实参传入线性回归模型 LinearRegression 的 fit 方法中，最终得到时长与成绩的线性回归方程。

12.4　数据可视化

现在我们来到数据分析的最后一个阶段——数据可视化。在前面的工作中，我们已经完成了数据的读取、数据的统计分析，并得出了一定的结论，但都仅限于在文字层面上的理解，未能用直观的图像、图形等展示形式支撑我们的结论，下面我们将基于 Python 的 matplotlib 模块，绘制以下数据图形：

（1）成绩排名柱状图。

（2）成绩分布直方图。

（3）成绩分布箱线图。

（4）成绩占比饼图。

12.4.1　成绩排名柱状图

在学生成绩表 12.1 中，记录了学生各科成绩的信息，现在我们在此基础上统计综合成绩，并以柱状图的形式展示其排名情况，具体程序如程序 12.11 所示。

程序 12.11　绘制成绩排名柱状图：

```
1:  import pandas as pd
2:  import matplotlib.pyplot as plt
3:  file_path = "report.csv"
4:  df_raw = pd.read_csv(file_path, encoding='utf-8')
5:  df_report = pd.DataFrame(df_raw.iloc[:, 3:].sum(axis=1), columns=["Report"])
6:  df_aggr = pd.concat((df_raw, df_report), axis=1)
7:  df_aggr_sort = df_aggr.sort_values(by=['Report'], ascending=False)
8:  width = 0.5
9:  fig, ax = plt.subplots()
10: ax.bar(df_aggr_sort["Name"], df_aggr_sort["Chinese"], width=width, label="Chinese")
11: ax.bar(df_aggr_sort["Name"], df_aggr_sort["Math"],
            bottom=df_aggr_sort["Chinese"],    width=width, label="Math")
12: ax.bar(df_aggr_sort["Name"], df_aggr_sort["English"],
            bottom=df_aggr_sort["Chinese"] + df_aggr_sort["Math"],
            width=width, label="English")
13: ax.set_xlabel("Name")
14: ax.set_ylabel("Report")
15: ax.legend()
16: plt.show()
```

分析：

在绘制具体图形前，需要准备好显示数据，在 matplotlib 模块中一般是以坐标点的形式，即全部数据的横坐标，纵坐标分别作为一个列表，传入绘图函数，如果要绘制学生成绩排名的柱状图，则需事先准备成绩排名的具体数据，因此我们首先将数据导入程序，统计学生综合成绩，并将其拼接到原始数据表中，之后就可按照综合成绩实现学生排名，如程序的第 5～7 行所示。在得到排名数据后，我们初始化绘图板，设置学生姓名列表为 x，各科成绩为 y，依次输入绘图函数 bar 中，最后通过调用 matplotlib.pyplot 的 show 方法，显示排

名图像，如图 12.1 所示。

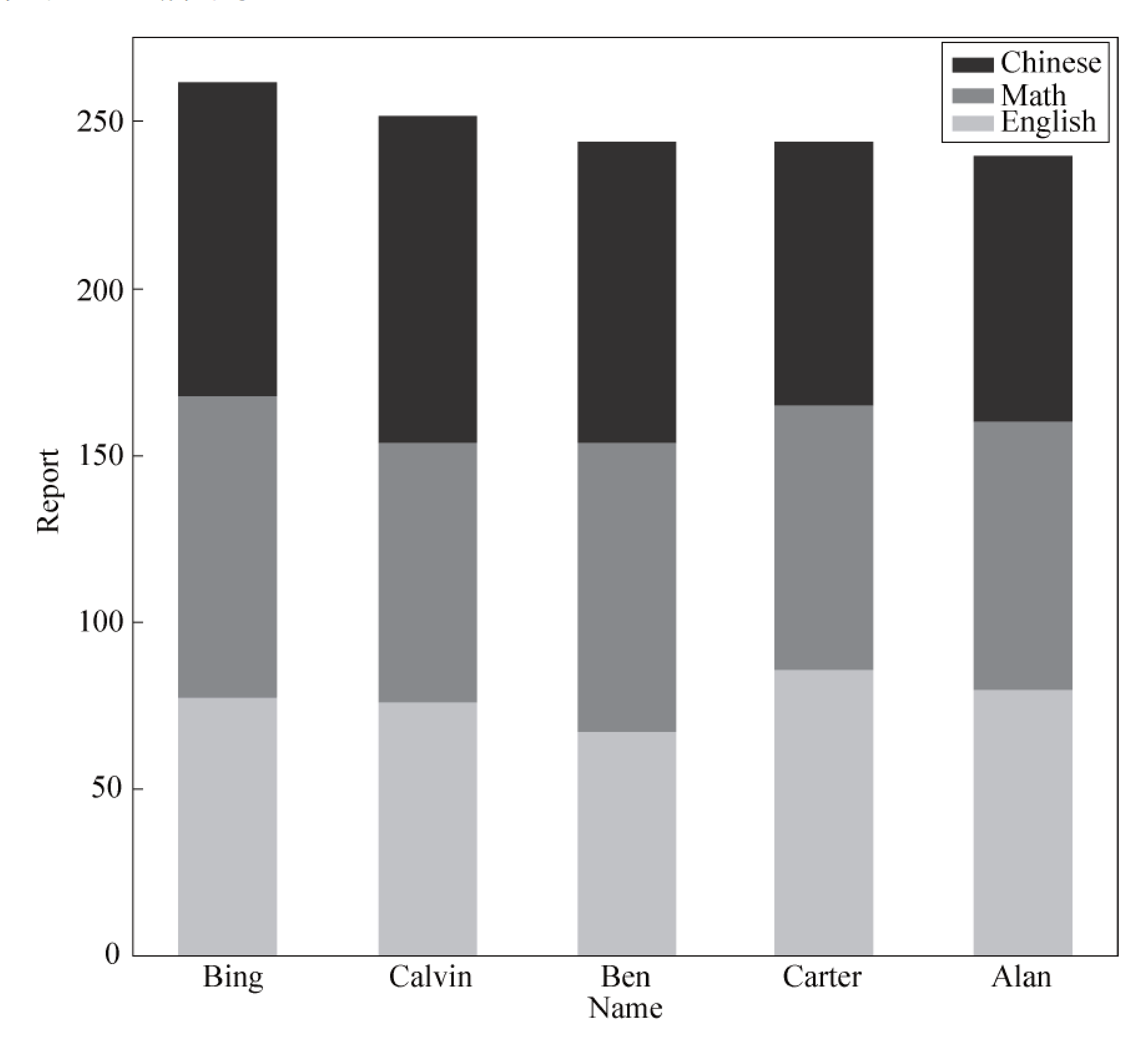


图 12.1　成绩排名柱状图

12.4.2　成绩分布直方图

为了客观了解成绩的整体情况，我们需要进一步统计成绩数据，并以直方图的形式呈现其分布状况，具体程序如程序 12.12 所示。

程序 12.12　绘制成绩分布直方图：

```
import pandas as pd
import matplotlib.pyplot as plt
file_path = "report.csv"
df_raw = pd.read_csv(file_path, encoding='utf-8')
df_report = pd.DataFrame(df_raw.iloc[:, 3:].sum(axis=1), columns=["Report"])
num_bins = 50
```

```
fig, ax = plt.subplots()
plt.hist(df_report["Report"], num_bins, normed=1, alpha=1)
ax.set_xlabel("Report")
ax.set_ylabel("Count")
plt.show()
```

分析：

我们在统计成绩的过程中，只需要计算每个学生的综合成绩，之后将综合成绩作为一维列表传入绘图函数 hist 中，在该函数内部会自动完成统计工作，最后我们调用 matplotlib.pyplot 的 show 方法，显示成绩分布直方图，如图 12.2 所示。

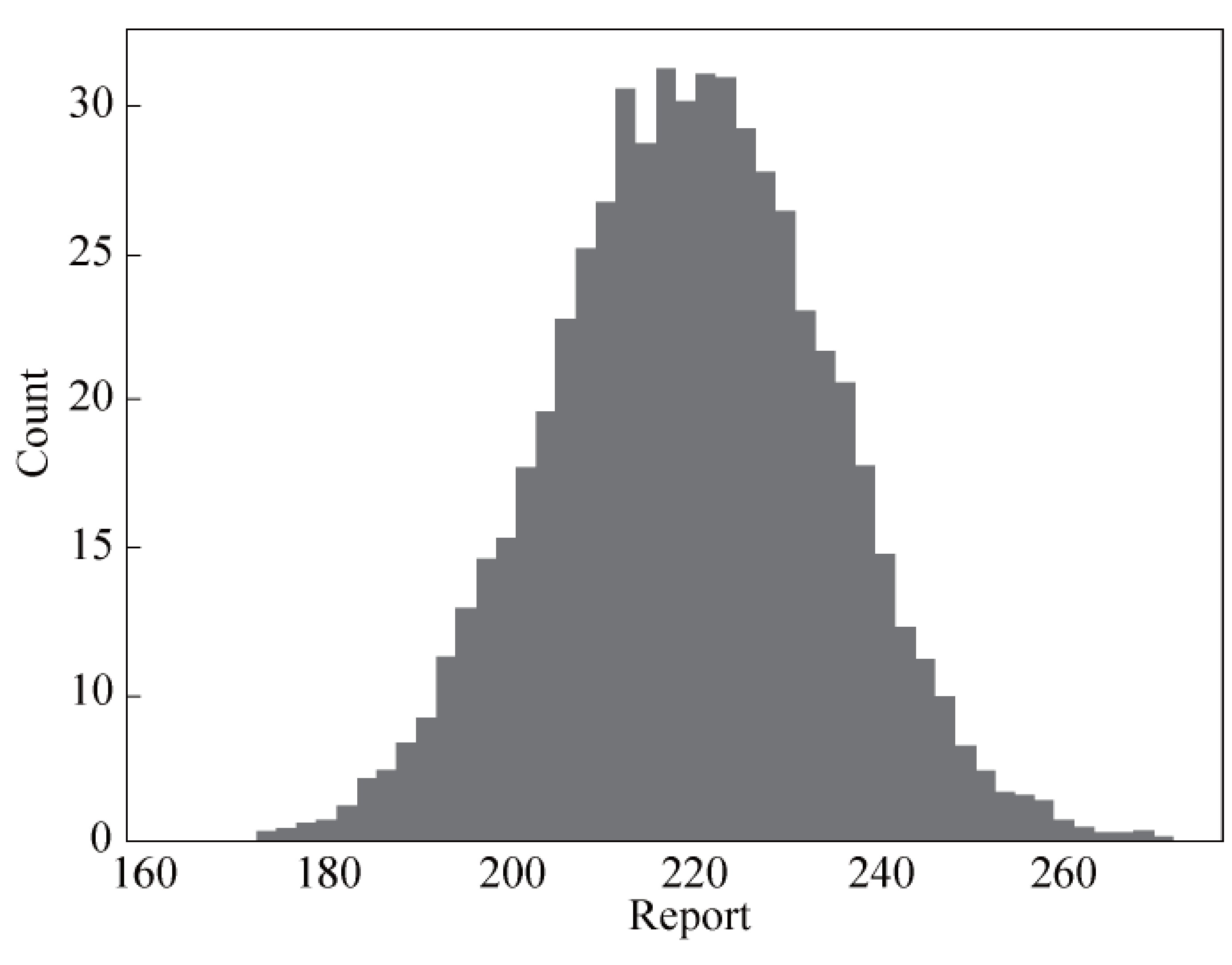


图 12.2　成绩分布直方图

12.4.3　成绩分布箱线图

还可以用另一种图形——箱线图来显示学生成绩的分布状况，它是一种用作显示一组数据分散情况资料的统计图，箱线图可以让我们直观地了解数据的中位数、异常值、分布区间等形状信息，下面我们先将图形绘制出来，再具体介绍其基本概念，绘图程序如程序 12.13 所示。

程序 12.13　绘制成绩分布箱线图：

```
import pandas as pd
import matplotlib.pyplot as plt
file_path = "report.csv"
```

```
df_raw = pd.read_csv(file_path, encoding='utf-8')
df_report = pd.DataFrame(df_raw.iloc[:, 3:].sum(axis=1), columns=["Report"])
fig, ax = plt.subplots()
plt.boxplot(df_report["Report"])
ax.set_xticklabels(["Report"])
ax.set_ylabel("Count")
plt.show()
```

分析：

与绘制直方图的过程类似，我们在统计完学生的综合成绩后，调用绘图函数 boxplot 将综合成绩数据作为实参传入，并通过 matplotlib.pyplot 的 show 方法显示成绩分布箱线图，如图 12.3 所示。其中，箱线图中引入了统计学的四分位数的概念，所谓四分位数，就是把组中所有数据由小到大排列并分成四等份，处于三个分割点位置的数字就是四分位数：

（1）第一四分位数（Q1），又称为“下四分位数”，等于该样本中所有数值由小到大排列后第 25%的数字，如图 12.3 中的矩形的底边边框。

（2）第二四分位数（Q2），又称为“中位数”，等于该样本中所有数值由小到大排列后第 50%的数字，如图 12.3 中的矩形中的内嵌线段。

（3）第三四分位数（Q3），又称为“上四分位数”，等于该样本中所有数值由小到大排列后第 75%的数字，如图 12.3 中的矩形的顶边边框。

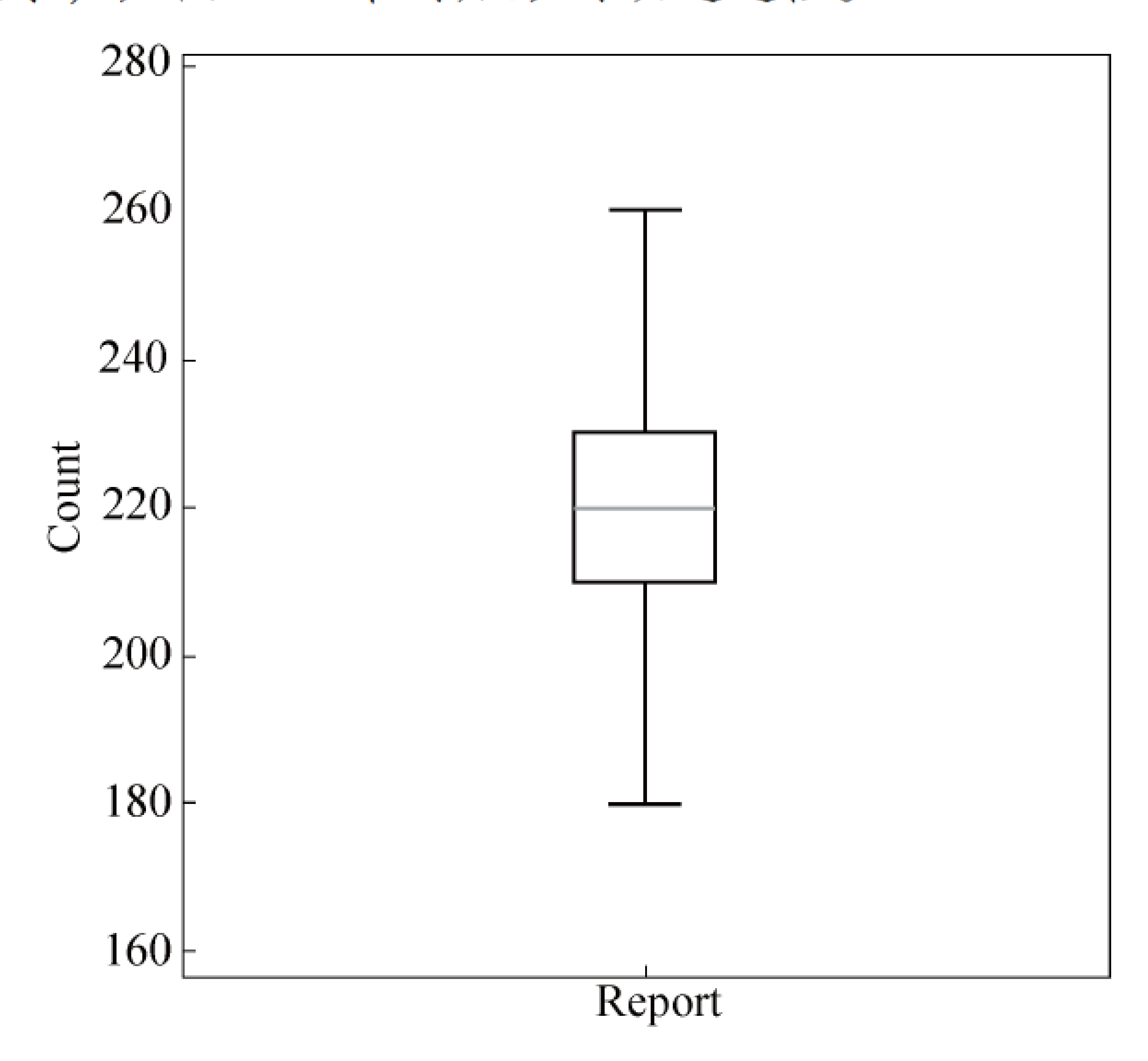


图 12.3　成绩分布箱线图

12.4.4　成绩占比饼图

在上述图形实例中，客观地反映了学生成绩的排名、分布等信息，但除此之外，我们还需要清楚地了解每个区间的成绩在整个成绩统计中的占比，以清晰反映与比较各区间成绩比重。下面将以饼图的形式将各个区间的成绩数据占比直观地呈现出来，具体程序如程序 12.14 所示。

程序 12.14　绘制成绩占比饼图：

```
from collections import defaultdict
import math
import pandas as pd
import matplotlib.pyplot as plt
file_path = "report.csv"
df_raw = pd.read_csv(file_path, encoding='utf-8')
list_report = list(df_raw.iloc[:, 3:].sum(axis=1))
num_bins = 5
report_max = int(math.ceil(max(list_report)))
report_min = int(math.floor(min(list_report)))
span = int((report_max - report_min) / num_bins)
threshold = [thred for thred in range(report_min, report_max, span)]
threshold_interval = defaultdict(lambda: 0)
list_interval = list(map(lambda v1, v2: (v1, v2), threshold[:-1], threshold[1:]))
for item in list_report:
    for interval in list_interval:
        if interval[0] <= item < interval[1]:
            threshold_interval[interval] += 1
            break
labels = []
counts = []
for label, count in threshold_interval.items():
    labels.append(label)
    counts.append(count / len(list_report))
explode = (0.1, 0, 0, 0, 0)
fig, ax = plt.subplots()
plt.pie(counts, labels=labels, explode=explode, autopct='%1.1f%%')
plt.show()
```

分析：

在绘制饼图时，同样需要准备数据，绘图函数 pie 接收各区间成绩的比例值，因此我

们需要自己划分区间，并为各区间成绩计数，最后计算各区间计数在成绩总数中的比重，如程序的第 8～24 行所示，我们首先通过成绩中的最大、最小值确定区间跨度 span，进而确定每个区间的开始阈值 threshold，并通过 map 函数组合 threshold 生成区间集合 list_interval，里面记录了每个区间的上下阈值；接着我们遍历成绩数据表 list_report，将每个成绩划入对应区间内，即对对应区间计数，之后将成绩区间与对应的计算结果放入 labels、counts 列表中，并将其作为实参传入 pie 函数中，最后通过 matplotlib.pyplot 的 show 方法显示成绩占比饼图，如图 12.4 所示。

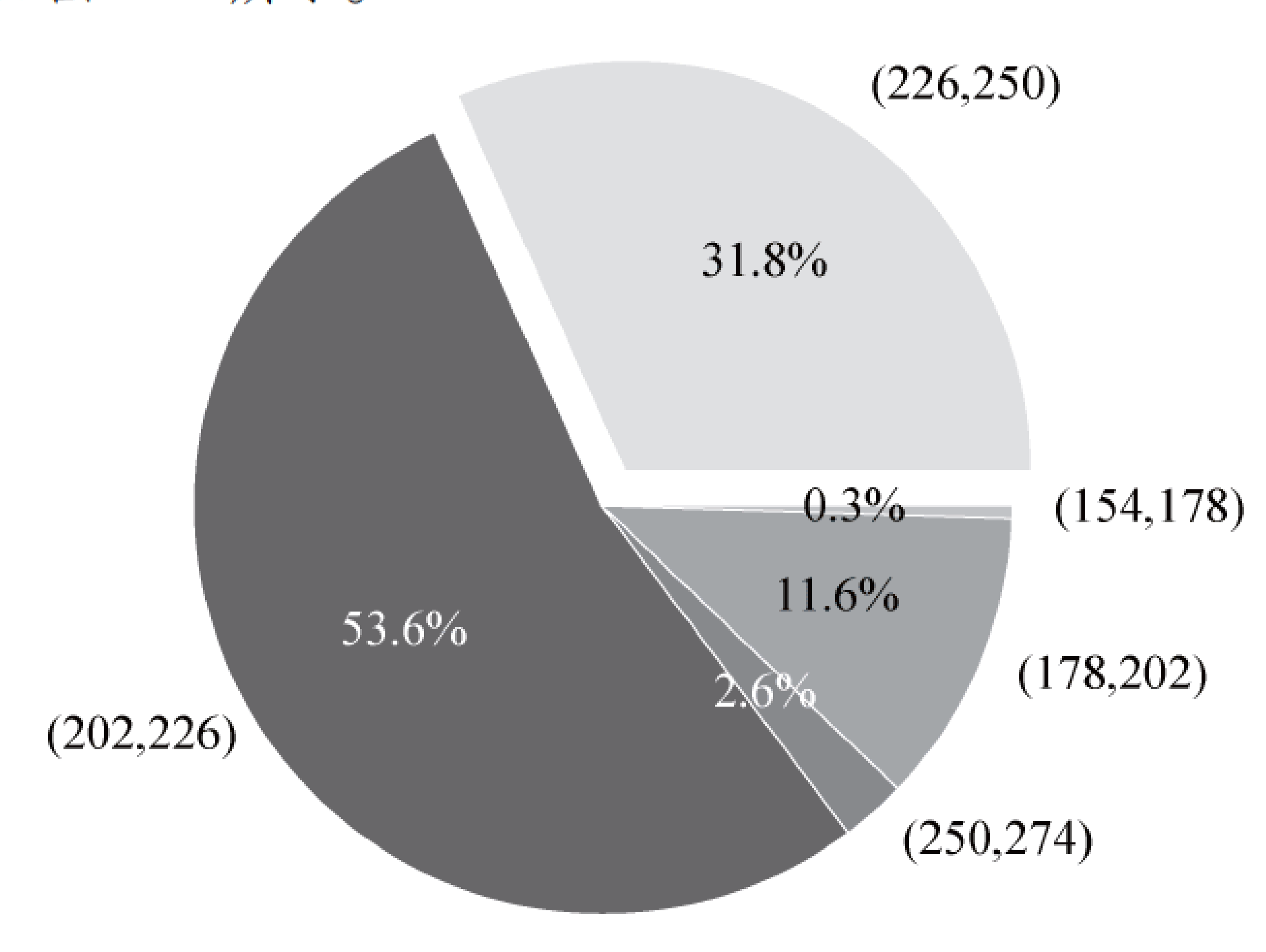


图 12.4　成绩占比饼图

12.5　总　　结

在本章中，首先通过介绍 pandas 模块的各种数据源的导入、导出方式，学习了基于 Python 的数据读取与转化。然后讲解了数据分析的过程，通过学生成绩表的案例回顾了统计学中的资料描述统计量、协方差、相关系数与线性回归分析等，对学习时长与学生成绩等维度的数据进行分析，建立两者的关联模型，发掘学习过程中的一般规律。

数据的可视化分析可让读者快速、直观地了解数据特征，我们学习了基于 Python 的 matplotlib 模块的常见图形绘制方式，并通过绘制图形进一步熟悉了对 pandas、matplotlib 及 numpy 等模块的函数调用。

12.6　练　　习

（1）什么是数据分析？数据分析常用的理论与方法又有哪些？

（2）参考 12.2 节的程序 12.2，完成对下列数据的导出到.csv 格式文件中，请完成程序的设计并写出其程序代码。

假设有高三某班各科平均分成绩如下：

语文	数学	英语	物理	化学	生物
110.8	115.3	105.24	89.80	87.31	78.05

要求：生成索引和表头。

（3）参考 12 章中的程序，将以下数据存为.xlsx 格式的 Excel 表格文件，然后完成下列练习：

	Student ID	Name	Hour	Chinese	Math	English
1	1807022101	Stu1	8	80	80	80
2	1807022102	Stu2	10	78	90	94
3	1807022103	Stu3	9	67	87	90
4	1807022104	Stu4	6	76	78	98
5	1807022105	Stu5	7	87	78	79
6	1807022106	Stu6	8	86	77	89
7	1807022107	Stu7	4	65	78	75
8	1807022108	Stu8	6	90	92	72

①写出对以上数据的 Excel 表格文件的导入程序。

②将 Excel 文件转为 CSV 文件，然后写出程序分析学生每天的学习用时与本次综合成绩的关联关系。

③基于②题的 CSV 文件统计综合成绩，并以柱状图的形式展示其排名情况。

④基于②题的 CSV 文件，编写程序并绘制成绩排名柱状图。

第 2 篇



人工智能篇

本篇将介绍人工智能的概念以及相关基础知识。考虑到高中生的数学基础和各个区域、各个学校的教学水平差异，我们提供了机器学习初步、自然语言处理、语音识别、计算机视觉和人工神经网络五部分的内容。

每部分的内容力图说清楚概念，讲清楚应用，实现一个案例。通过学习，我们期待学生可以了解基础知识，懂得基础原理，实践一个案例。这样做到知行合一，以实践带动知识的理解，将现实世界的实践和书本的知识融合起来。

第 13 章　人工智能导论

通过本章，我们将充分了解到人工智能的一些基本概念和基本内容。本章将要学习：

- 人工智能定义及发展。
- 高中阶段为什么学习人工智能。
- 人工智能的分支。
- 加速回报定律。
- 人工智能与伦理。
- 图灵测试。
- 人工智能与机器人。

13.1　人 工 智 能

什么是智能？世界上所有的东西都是智能的吗？假如，世界万物都是智能的，那我们怎么意识到“智能”的存在？我们将如何区分智能和非智能呢？所以，在介绍人工智能之前，有必要先给出“智能”的定义。“智能”简单来说是让机器有“目的性”地“思考”。

我们思考一下，计算机显示器屏幕上显示的内容有目的性，那么计算机就是智能的吗？答案肯定不是啦！因为“目的”的主体是智能体，计算机显示器屏幕上显示内容的“目的”是人的“目的”，而不是计算机的“目的”。所以，计算机显示器屏幕上显示内容是人的智能，而不能说计算机是智能的。这就表明“智能”涉及其他诸如意识、自我、思维等。既然已经懂得了“智能”的含义。下面，我们来看看什么是人工智能。“人工”一词比较好理解。“人工”即人力制造的，或者说人们自身的智能程度已经达到可以创造智能的地步。

13.1.1　什么是人工智能

人工智能（Artificial Intelligence，AI）是一门综合了计算机科学、生理学、哲学的交

叉学科，是让生物的自然智能在计算机上得以实现，重在模拟人的思维过程和智能行为的学科，同时人工智能在计算机领域内越来越受关注。

13.1.2　强人工智能与弱人工智能

人工智能分为强人工智能和弱人工智能，它们是用来评价人工智能的能力，不是判断人工智能有无作为的标准，而是就人工智能如何思考来明确各自的立场。

“强人工智能”一词最初是约翰•罗杰斯•希尔勒（见图 13.1）提出来的，其定义为“计算机不仅是用来研究人的思维的一种工具，相反，只要运行适当的程序，计算机本身就是有思维的”。但是，对于约翰•罗杰斯•希尔勒本人，他完全不相信计算机真的可以像人类一样思考。对于上述定义，只是他认为“强人工智能群体”是这样想的，计算机本身不可能具有自己的思维。现在，“强人工智能”指的是机器可以根据情况对事物进行推理并且能有效地解决问题，这样的机器可以被认为是有知觉，有自我意识的，并且机器能独立解决问题，找到最优方案。强人工智能可以分为两类：类人的人工智能和非类人的人工智能。前者指的是机器的思维和推理就像人一样，后者指的是机器产生了和人完全不一样的认知，有和人类完全不一样的推理方式。



图 13.1　约翰•罗杰斯•希尔勒

弱人工智能坚持认为人类不可能制造出有自己的思维并且能真正地推理和解决问题的智能机器。这些机器仅仅是看起来智能，但并不是真正的智能，更不会有自我意识，目前的主流思想也集中在弱人工智能上。

13.1.3　人工智能的发展史

1942 年美国科幻巨匠阿西莫夫提出“机器人三定律”，该定律为后代创作提供了巨大的指导意义。1956 年夏天，美国达特茅斯学院举办的达特茅斯会议（见图 13.2），是历史上第一次人工智能研讨会，人工智能首次在会议上被提出来。这次会议也被认为是人工智能诞生的标志。在会上，麦卡锡提出了“人工智能”概念，纽厄尔和西蒙则展示了编写的逻辑理论机器。



图 13.2　达特茅斯会议

1942 年，阿西莫夫在他的小说《我，机器人》中，提出“机器人三定律”：

（1）机器人必须不危害人类，也不允许看到他人受伤害而袖手旁观。

（2）机器人必须绝对服从于人类，除非这种服从有害于人类。

（3）机器人必须保护自身不受伤害，除非为了保护人类或者是人类命令它做出牺牲。

1968 年—1972 年间，美国斯坦福国际研究所研制了移动式机器人 Shakey，这是首台采用了人工智能的移动机器人。它装备了电视摄像机、三角测距仪、碰撞传感器等，并通过无线通信系统由两台计算机控制，能进行简单的自主导航。

1970 年，维诺格拉德设计研发的 SHRDLU 系统，可以正确理解语言。1976 年美国斯坦福大学研发的 MYCIN 专家系统可以诊断传染性血液病患者。1981 年，日本经济产业省拨款 8.5 亿美元用以研发第五代计算机项目，人工智能计算机登场。随后英国、美国也开

始向计算机领域的研究提供大量资金。在 1987 年—1993 年，人工智能由于被认为并非下一个发展方向，拨款受到了限制。在失去资金支持后，人工智能遇到了低谷，但随后不久人工智能开始逆袭。1997 年，比尔 • 盖茨耗时七年，花费高达 9700 万美元的豪宅建成，这是第一座智能家居（见图 13.3）。

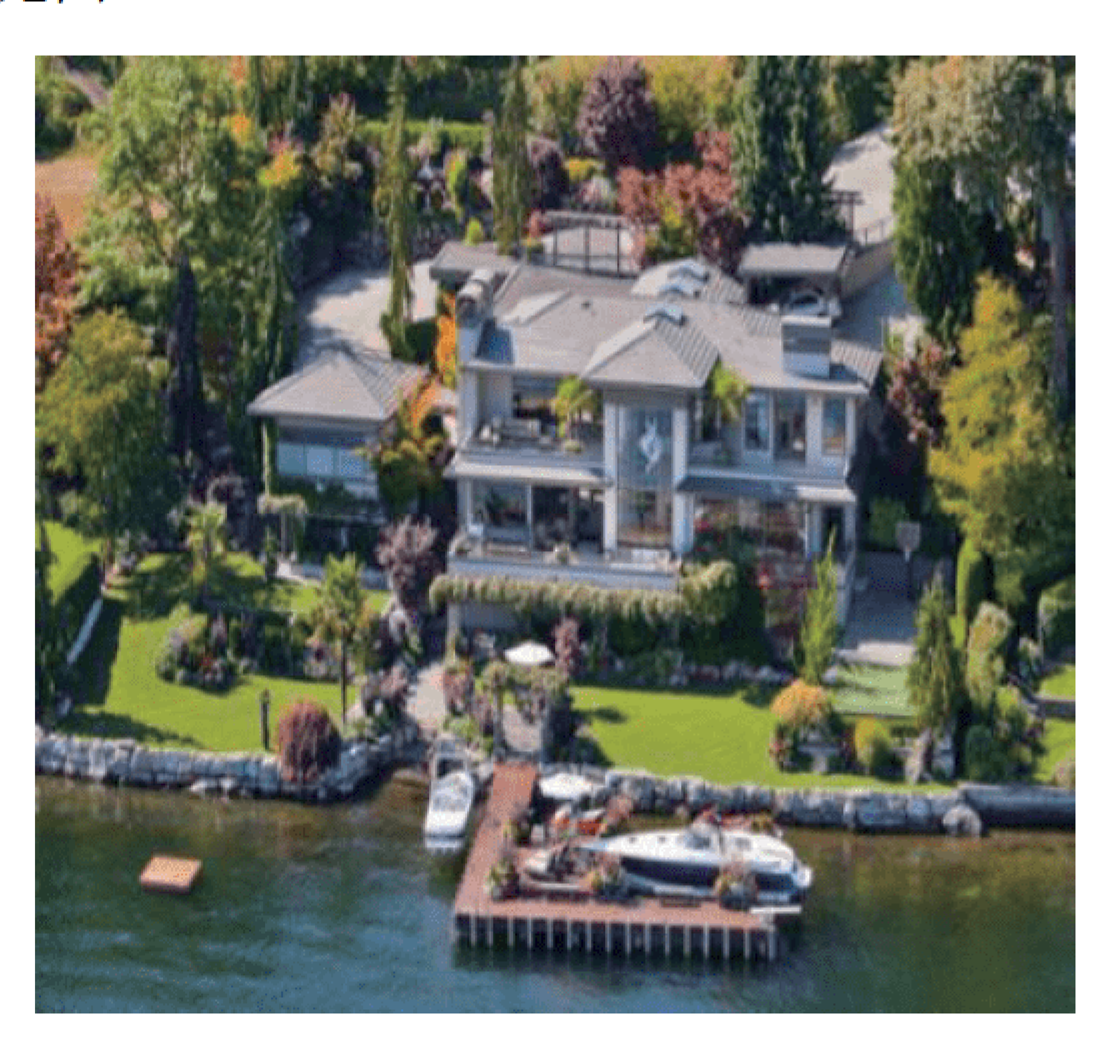


图 13.3　比尔 • 盖茨的豪宅

比尔 • 盖茨的这座豪宅完全按照智能住宅的概念建造，不仅具备高速上网的专线，所有的门窗、灯具、电器、池塘都能够通过计算机控制，所有设备都能自动调整，就连家中的一棵百年老树都有自己的传感器。不仅如此，家中还有一个高性能的服务器作为管理整个系统的后台。由于通信的自由化与高层次化，人类对工作环境的安全性、舒适性、效率性要求也大大提高，使家居智能化的需求变得格外重要。此外在科学技术方面，由于计算机控制技术和电子信息通信技术的发展，也促成了智能家居的诞生。

2014 年 5 月 28 日，在 Code Conference 大会上，谷歌（Google）推出了自己的新产品——无人驾驶汽车（见图 13.4）。与一般的汽车不同，谷歌无人驾驶汽车没有方向盘和刹车。这就意味着传统汽车的油门、方向盘、刹车等必不可少的配件，在无人驾驶汽车上已经全部看不到了，它主要依靠车内的以计算机系统为主的智能驾驶仪来实现无人驾驶。并且谷歌也希望无人驾驶汽车可以适应不同的场景，用户只需告诉车子他们的目的地，并且按一下按钮，车子就会自动带他们去那里。



图 13.4　无人驾驶汽车

13.2　为什么学习人工智能

就人工智能的本质而言，它是对人的思维过程的模拟，这主要包括两个方面。一是结构模拟，仿照人脑的结构机制，制造出“类人脑”的机器；二是功能模拟，暂时撇开人脑的内部结构，而从其功能过程进行模拟。2017 年 7 月 21 日，国务院新闻办公室举行国务院政策例行吹风会，重点介绍《新一代人工智能发展规划》。科学技术部副部长李萌指出，新一代人工智能具有五大特点，一是从人工知识表达到大数据驱动的知识学习技术；二是从分类型处理的多媒体数据转向跨媒体的认知、学习、推理，这里讲的“媒体”不是新闻媒体，而是界面或者环境；三是从追求智能机器到高水平的人机、脑机相互协同和融合；四是从聚焦个体智能到基于互联网和大数据的群体智能，它可以把很多人的智能集聚融合起来变成群体智能；五是从拟人化的机器人转向更加广阔的智能自主系统，如智能工厂、智能无人机系统等。

现在你一定很想问，作为高中生的我们，为什么要学习人工智能？现在学习人工智能会不会太早？其实，在 2018 年 1 月 16 日，教育部已经正式将人工智能、大数据、物联网、算法等加入了“新课标”的改革中。由于我国人工智能人才缺口大，而高中阶段精力充沛，

具备了初步的数学基础，是学习人工智能的最好阶段。这时学习人工智能，不仅能为自身的未来求学奠定基础，而且也为国家的人工智能发展和应用建立了雄厚的人才储备源。人工智能已经走进高中信息技术的新课标，可以在大学之前就提前培养人工智能人才。高中阶段所学的知识对学生的终身发展起到重要的作用，知识型内容与基本概念、基本原理的相关性高，时效性也长久，对学生终身学习和发展的价值也很大。牛津大学的卡尔•弗瑞与迈克尔•奥斯本发表的未来就业报告指出：未来几年，有 47%的工作有很大几率被人工智能取代，而人工智能及其编程的学习可以让孩子从容面对未来职业，更好地成就个人发展和社会发展。

13.3　人工智能的种类

人工智能主要分为两个种类，一是运用符号思考的人工智能，二是运用神经网络思考的人工智能。

运用符号思考的人工智能即符号主义（Symbolism），是一种基于逻辑推理的智能模拟方法，又称为逻辑主义（Logicism）。其原理主要为根据符号和规则来创造智能。一直以来，处在人工智能主导地位的便是符号主义。纽威尔和西蒙提出的“物理符号系统假设”为符号主义的实现打下了基础。该学派认为：人类认知和思维的基本单元是符号，而认知过程就是在符号表示上的一种运算。它把人看成一个物理符号系统，同时计算机也是一个物理符号系统。此时，我们就可以用计算机来模拟人的认知和行为。运用符号思考的人工智能实质在于模拟人的左脑逻辑思维，通过研究人类认知的原理，进而用符号来模拟人类的认知过程。

符号主义学派认为人工智能源于数学逻辑，并且认为功能模拟方法才应是人工智能的研究方法。通过分析人类认知系统的功能，用计算机模拟这些功能，进而实现人工智能。然而，符号主义主张用逻辑方法建立人工智能体系时，却遇到了“常识性”问题的障碍，对于那些不确定的事物的知识表示和难以表达的问题，符号主义不能提供好的解决办法，因此，受到其他学派的批评与否定。

运用神经网络思考的人工智能即人工神经网络，是一种针对人脑神经元网络进行抽象建立的简单模型，它按照不同的链接方式进而组成不同的网络（见图 13.5）。神经网络是一种运算模型，由大量的神经元相互链接而成。在最近的十几年来，人工神经网络取得了

非常大的进步。主要应用于模式识别、自动控制、生物、医学等领域。

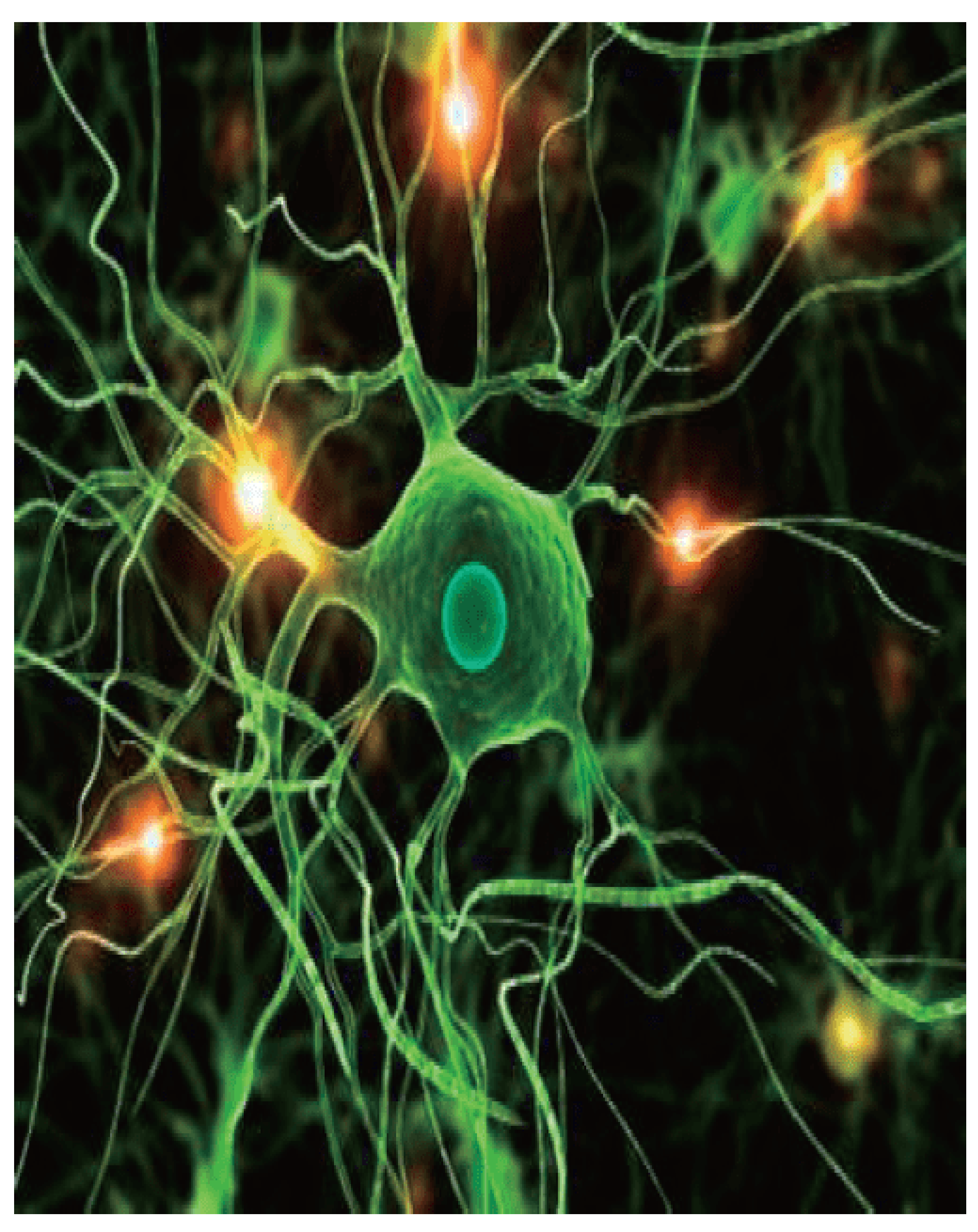


图 13.5　神经突触

人工神经网络的基本特点如下。

（1）具有自学习能力：例如，图像识别，先把不同的图像样板和素材输入到人工神经网络，此时网络就会通过自己的自学能力，渐渐学会分析识别相似的图片。这种自学能力对于预测功能有重大的帮助。预测人工神经网络也为经济、市场、效益等方面，提供远大的前景。

（2）联想存储功能：记忆并不像计算机里的存储器，记忆只是训练后，各个神经网络的权值和偏置。记忆数据已经固化到一个具有某功能的神经网络结构中。这整个被训练好的神经网络，就是记忆。人脑记忆不能离开神经网络单独存在。要移植记忆，就要重构神经网络，不像下载到计算机硬盘那么简单，它是功能性的。例如，一个叫小李的人的手机号码。小李→1→3→5→4→6→4→2→2→1→9→1，小李是第一个神经元的输入，然后输出是 1，1 是下一个神经元的输入，然后 3 是输出，以此类推，如此便形成了联想链。

（3）快速寻找优化解的能力：当我们在找一个复杂问题的解决方案时，总会需要大量

的计算，为此，设计一个针对特定问题的人工神经网络，能更加有效地提高运算能力，更快地找到问题的最优解决方案。

虽然人工神经网络的理论和算法还需要进一步的改进和完善。但是由于它的学习规则简单，可以轻松地在计算机上实现，并且有强大的记忆能力和自学能力，使其在市场上得到了更多的应用。

13.4　人工智能的分支

通过人工智能，可以学习到多个领域的知识。人工智能的研究方向已经被分成几个子领域，以下列出人工智能中重要的一些话题。

13.4.1　机器学习

机器学习，即机器本身在学习，是在研究计算机怎样模拟或实现人类的学习行为，使它学习到新的知识和技能。机器通过已有的知识结构，将它们重新组织，进而不断完善自身的性能。机器学习的应用十分广泛，例如，大家熟知的搜索引擎、医学诊断、语音和手写识别、战略游戏等领域。机器学习现在也成为数据分析领域的一个热点，在大多数人的平时工作中都或多或少涉及机器学习算法。现在我们来简单介绍下机器学习的 3 种学习方式：监督学习、无监督学习和半监督学习。

（1）监督学习：指的是建立基于标签训练数据的机器学习模型的过程，将某次输入应该采取的行动作为指令数据。举一个例子来说明，在虚拟空间内，制作一只机器狗，我们对狗发出“站”“转圈”“坐”的指令。虚拟的机器狗最初不知道每个指令应该对应什么动作，它就会任意地做动作。但当它做完动作之后受到鼓励时，就会在下次遇到这个指令时，做同样的动作。如果受到了训斥，就不会做这样的动作。经过这样循环往复的训练，机器狗就会记住什么样的指令对应什么动作。

（2）无监督学习：无监督学习是不需要标签训练数据的机器学习。2016 年谷歌旗下的 DeepMind 公司的戴维 • 西尔弗、艾佳 • 黄和戴密 • 哈萨比斯与他们的团队开发的 AlphaGo，就是通过无监督学习实现的。AlphaGo 是一款围棋的人工智能程序，通过自我对战的无监督学习，从开始的什么都不会，在短短三天内，就成为顶级的高手，打败了许

多围棋冠军。通常来讲，监督学习需要很多的数据，而无监督学习主要需要好的学习环境。

（3）半监督学习：是监督学习与无监督学习相结合的一种学习方式。意在利用少量的标签进行训练和分类学习。这种学习方式可以用来进行预测，通过应用于分类和递归等地方，对未标识的数据进行建模，在此基础上对标识的数据进行预测。

13.4.2　模式识别

模式识别是一门交叉学科，源于自动控制与计算机技术。同时又和人工生命、机电等学科紧密联系。模式识别是对事物或现象的各种信息进行处理分析，同时对这些事物或者现象进行分析解释的过程。例如，汽车车牌号的辨识涉及图像处理分析等技术。简言之，模式识别能让计算机认识到周围的事物，使人类与计算机能更自然方便地沟通。模式识别包括文字识别、语音识别、自然语言理解、计算机图形识别等。下面我们来重点了解一下文字识别和语音识别。

（1）文字识别：一般包括信息的采集、分析、处理及分类判断等过程，是一种计算机自动识别字符的技术，也是模式识别的一个重要的应用领域。汉字已经承载了数千年的历史，同时也是使用人数最多的文字。我们在日常的生产和生活中，要处理大量的文本、图标，这些事情琐碎又麻烦。为了减轻人们的劳动，将文字方便、快速地输入到计算机中，已经成为现如今一个重要的问题。目前，汉字的输入主要分为人工键盘输入和机器自动识别输入两种。人工键盘输入是最常见的一种，速度慢而且劳动强度也很大。机器自动识别输入又分为汉字识别输入和语音识别输入。从技术角度来看，识别手写体的难度高于印刷体，在识别手写体时，脱机手写体的难度远超于联机手写体识别。

（2）语音识别：在近二十多年来，语音识别取得了非常大的进步，已经从实验室走入了市场。其目标是将人类语音中的词汇转成计算机可读的输入。语音识别技术主要包括语音拨号、语音文档检索、听写数据的录入、语音导航等。所涉及的领域有信号处理、发声机理和听觉机理等。近年来，语音识别在移动端的应用也非常火热，许多公司都对其投入了大量的人力和物力。例如，国内的科大讯飞语音助手、百度语音等系统都采用了最新的语音识别技术。语音识别也会日益成为人们日常生活和工作中特别重要的技术。

模式识别的研究主要集中在两个方面，一是在特定的条件下，怎样让计算机实现模式识别的理论和方法；二是研究人类和其他一些生物体是怎样感知对象的，这一点属于认知

科学的范围，是生理学家、生物学家和神经学科学者的研究内容。而第一点主要是数学家、计算机科学研究人员研究的内容，并且他们经过几十年的不懈努力，已经取得了系统的研究成果。模式识别可用于文字和语音识别、遥感和医学诊断等方面。

13.4.3　知识表示

知识表示是人工智能最基础的概念，是指对人工智能的知识形态的表述。知识表示把知识客体中的知识因子与知识相关联，有利于人们的识别和理解。知识表示是知识组织的前提和基础，任何知识组织方法都是要建立在知识表示的基础上。谓词逻辑是一种比较常见的知识表示方法。在谓词逻辑中，命题是用谓词表示的。谓词的一般形式是 $p(x_1, x_2, \ldots, x_n)$，其中 P 是谓词名称，$x_1, x_2, \ldots, x_n$ 是个体。举一个例子，让我们更好地理解知识表示中的谓词逻辑。

假设有这样一个知识需要表示：小赵是音乐系的学生，但他不喜欢唱歌。我们用一阶谓词逻辑来表示它就需要采用如下的步骤。

首先，定义谓词如下：

$Music(x): x$ 是音乐系的学生。

$Like(x, y): x$ 喜欢 y。

那么，我们可以使用谓词公式表示如下：

$$Music(xiaozhao) \wedge \neg Like(xiaozhao，sing)$$

1. 启发式算法

启发式算法是一个由直观或经验而构造的算法，指在可以接受的花费下得出待解决问题的每一个可行解，这个可行解与最优解的偏离程度一般情况下是不能被预计的。简单来说，启发式算法在解决问题时，利用过去的经验，选择确定是有效的方法，而不是以确定的步骤去找答案。它最大的优点是在有限的搜索空间内，很大程度上减少了尝试的次数，缺点是这种方法有失败的可能性。

2. 遗传算法

遗传算法是模拟达尔文进化论的人工智能，借鉴生物界的进化规律（适者生存，优胜劣汰的遗传机制）演化而来的随机化搜索方法。它有对结构对象进行操作的特点，很好的全局寻找最优解的能力，能够自动优化搜索空间，并且不依赖梯度信息或其他辅助知识。

遗传算法被广泛地应用于机器学习、信号处理、组合优化等领域，是现代关于智能计算的一种关键技术。

在 20 世纪 90 年代，遗传算法达到兴盛时期，在理论研究和应用领域都十分受欢迎。渐渐由于应用领域的扩展，遗传算法有了五个引人注目的新方向，此处只需大概了解。一是基于遗传算法的机器学习，二是应用于神经网络的遗传算法，三是并行处理的遗传算法，四是遗传算法与人工生命的研究，五是遗传算法与进化规划。

遗传算法的性质，已被人们广泛地应用于机器学习、信号处理、组合优化、自适应控制和人工生命等领域。它是现代有关智能计算中的关键技术。遗传算法也是计算机科学人工智能领域中用于解决最优化的一种搜索启发式算法，是进化算法的一种。这种启发式算法通常用来生成有用的解决方案来优化和搜索问题。

3. 深度学习

深度学习是一种新兴的“超能力”，可以用于搭建自己的人工智能系统。深度学习源于神经网络，它通过组合底层特征形成抽象的高层表示属性特征，用以发现数据的分布式特征表示。深度学习通过模仿人脑的机制来解释数据，应用于语音识别、自然语言处理、计算机视觉等其他商业领域，深度学习属于无监督学习的一种。下面简单了解深度学习的一些应用程序。

盲人看图片：这是 Facebook 的一项推新技术，该技术利用人工智能的深度学习，让有视力障碍的人也能看照片。此项技术通过深度学习，在盲人阅读新闻时会自动生成相关图片的文字版，同时将文字以语音的形式输出，让盲人和视力障碍人士也能在 Facebook 上阅读新闻和看图片。

语音实时翻译：被誉为“全球十大突破性技术”。随着国际文化的飞速发展，人们对翻译有更高的要求，语音实时翻译打破了各国语言不通的障碍，使语言沟通的实时性不再是问题。而这一项技术最大的意义在于，使人们可以无障碍地进行信息交流的愿望得以实现。

自动回复电子邮件：2015 年，谷歌推出有别于过去的邮箱“自动回复”功能，这一项技术通过深度学习得以实现。该功能实现了系统在收到邮件后，判断该邮件是否需要及时回复，同时可以提出三个合适的候选答案，用户可以直接单击发送，也可以编辑之后再进行发送。当用户手动回复之后，该系统会自动记录类似问题的最优回复，以备下次使用。

13.5　加速回报定律

加速回报定律是信息科技中的基本理论，它遵循可预见的指数级增长规律，反对传统的“无法预知未来”的观念。虽然仍有许多事情都是未知数，但事实证明，基础性价比及信息承载量却确确实实可以预见。更让人吃惊的是，这些变化并不受战争或和平、繁荣或萧条等因素的干扰。

想象一下，假如现在是 2045 年，你坐着时光机回到了 1790 年，那个时代还没有电，交通靠步行或者马车。人们之间的来往需要靠书信，从北京到四川如果是策马奔腾的话要 50 多个小时，如果使用马车或者步行，可能长达数日或者几十日。你在那个时代邀请了一位叫小杨的朋友来 2045 年玩，让他感受一下“未来”。你带着他像大鹏一样在天空中翱翔，在深海中畅游，与大西洋另一头的人在聊天。你还没有跟他解释过网络、机器人、核武器是什么东西。这时候小杨会是什么体验？应该可以用目瞪口呆来形容了。现在，如果小杨回到了 1790 年，他觉得自己的经历很囧，也想体验一下把别人吓到的感觉，于是他也回到了相差 255 年前的 1535 年，邀请生活在 1535 年的小寇来 1790 年玩，小寇可能会对 255 年后的生活感到震惊，但至少不会目瞪口呆，因为 1790 年和 1535 年的生活相差并不是很大，如果小杨想把别人吓到，估计要回到公元前才能满足他的愿望。

这样讲来，你是否理解了加速回报定律了呢？那么你觉得“2050 年世界会变得面目全非”这句话可笑吗？人类的发展如图 13.6 所示。

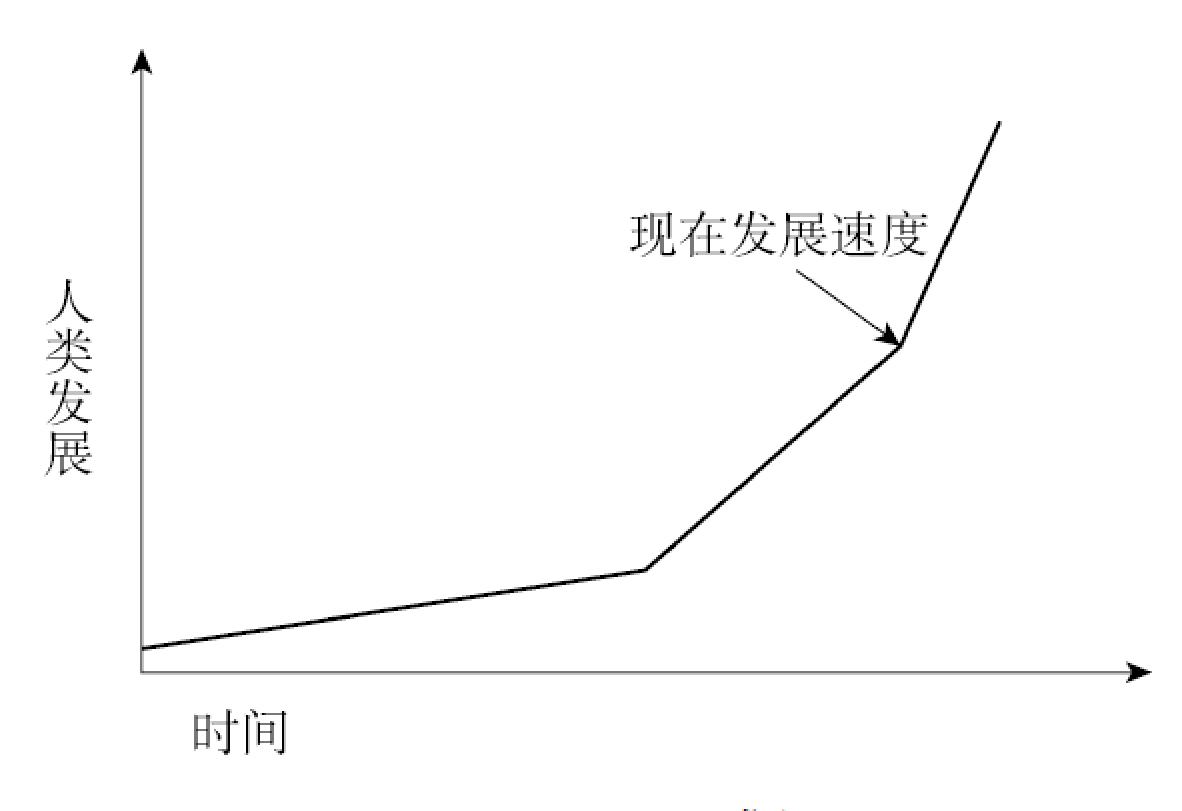


图 13.6　人类的发展

其次，你可能听过“奇点”或者“技术奇点”这种说法。这种说法在数学上用来描述类似渐进的情况，这种情况下通常的规律就不适用了。这种说法同样被用在物理上来描述无限小的高密度黑洞，同样是通常的规律不适用的情况。Ray Kurzweil（雷•库兹韦尔）则把奇点定义为加速回报定律达到了极限，技术进步以近乎无限的速度发展，而奇点之后我们将在一个完全不同的世界生活。

13.6　人工智能与伦理

人工智能要重视伦理价值。人们对人工智能的关注，也是对人类自身的关注。人工智能技术越来越发达，人类会被取代吗？或者说人类会成为人工智能的奴隶吗？所以人工智能与伦理问题理应得到广泛的重视。

虽然人工智能的持续进步和广泛应用带来的好处是巨大的，但是，为了让人工智能真正有益于人类社会，伦理问题应受到更多关注。首先，就是隐私问题。现如今的人工智能系统，基本都是对大数据的学习，此时需要大量的数据。一方面这其中会使用很多的敏感数据，这些数据会被泄露出去，将对个人的隐私产生威胁。其次，关于人类道德责任方面的问题。例如，如果自动驾驶汽车、智能机器人对我们人类造成了人身伤害，那么该由谁承担这个后果？由于系统是自主的，开发者很难预测，意外事故又应该由谁来承担法律责任。最后就是机器人的权利。机器人越来越智能化，它们将在人类社会中扮演什么样的角色？它们在法律上是什么？自然人？法人？我们可以杀死它们吗？杀死它们会犯法吗？这些问题都是非常值得深思的。

所以，在这个人工智能快速发展的时代，对人工智能进行伦理测试同样重要，包括道德、隐私、正义、有益性、安全、责任等。

13.7　图 灵 测 试

图灵测试是测试人在与被测试者（一个人和一台机器）隔开的情况下，通过一些装置（如键盘）向被测试者随意提问。问过一些问题后，如果被测试者超过 30%的答复不能使测试人确认出哪个是人、哪个是机器的回答，那么这台机器就通过了测试，并且被认为具

有人类智能（见图 13.7）。

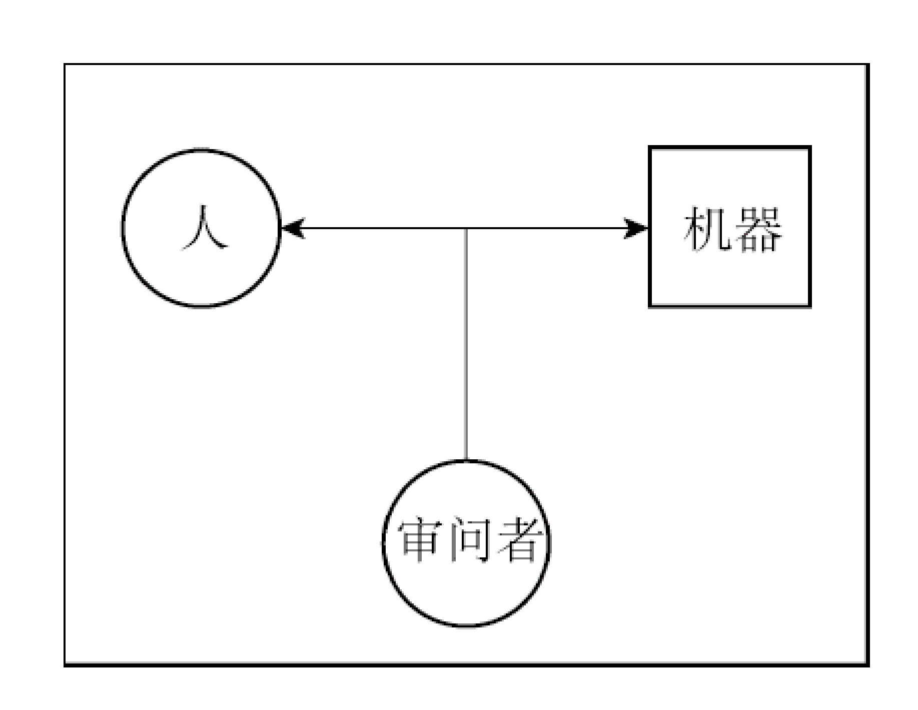


图 13.7　图灵测试

在 2018 年 Google I/O 大会中，CEO Sunder Pichai（桑达尔 • 皮查伊）直接拿出这次大会的王牌——Google Assistant，它便应用了图灵测试的原理，是一款完全颠覆人们认知的程序。相比较 Apple Siri 和 Microsoft Cortana 的机械式语音对答，Google Assistant 不仅能听懂我们说的话，同时还能进行无障碍的沟通。在这次发布会上，用户对 Google Assistant 说“我想理发”，Google Assistant 就会直接帮忙电话预约！而且整段和理发师的对话过程，表现得都无比自然流畅，理发师完全没有察觉到自己竟在和机器对话。Google Assistant 与真人对话时，也丝毫没有表现出任何的滞后或逻辑上的错误。它更加厉害的地方在于，通过学习，可以像朋友一样陪我们聊天。

13.8　人工智能与机器人

近几年，回顾人工智能走进校园的历程，我们很容易发现，人工智能进入学校的重要载体就是机器人了。2017 年 7 月 8 日，国务院出台《新一代人工智能发展计划》指出，推动人工智能在教学、管理、资源、建设等方面的应用。因此，将机器人用于教育中，尤其是国内外很多优质的高中学校已经开设了各种各样的机器人创客教育，将人工智能的应用推向了一个新的高度。

机器人是一个涵盖机械、电子、自动控制、软件等诸多学科的集成产品，当我们将人工智能的思想，诸如语音识别、视觉、自然语言处理等附加其上，那么一个具有人工智能的智能机器人就应运而生了。人工智能极大地促进了机器人产品的智能化，而机器人作为

人工智能的载体将为人们的生产生活提供更多的便利化服务。

本书及其配套实验教材的主要目的是希望学生可以通过我们自主开发的机器人产品来进行人工智能的学习，在学完本套教材后，学生可以将简单的案例写入机器人中，这样更能激发学生的学习兴趣，同时培养学生的综合能力。这些机器人尤为适应新课程，大大提高了学生的科学素养，学生可以在“学中玩，玩中学”，机器人将会像计算机一样遍及我们的校园，并且机器人也是人工智能学习很好的载体。机器人教学集中承载着信息技术教育的各种核心价值，具有很强的持续发展潜力。在国外，机器人教育一直是个热门话题，在 1994 年麻省理工学院为了提高计算机专业学生的设计能力，设立了“设计和建造 LEGO 机器人”的课程。1992 年开始，美国在高中生中就已经推行了“感知和认知移动机器人”。在我们国家，用于教育的机器人在许多省市都有很大的发展，在新的高中课程标准中也设立了“人工智能初步”的选修模块，成为人工智能教育的第一步。因为机器人刚进入课程，所以参与学科教学的经验并不多，虽然高中的课程中增加了人工智能这一部分，但是仍然是理论较重。所以此书中，我们在讲理论的同时，加重了实践这一方面，将人工智能融入机器人中，更有利于开展相关的教学，而且机器人中含有多种学科知识的技术，几乎是伴随着人工智能产生的。人工智能与机器人紧密联系，我们也相信，随着人工智能的不断发展，用于教育的机器人也将会更为全面。

13.9　人工智能与 Python

人工智能程序几乎可以使用所有的编程语言来实现，如 C/C++，Java 等，那么为什么首选 Python 呢？

Python 作为美国主流大学受欢迎的入门编程语言，诞生至今已经有 20 多年的时间，相对于其他编程语言，Python 更加易学、易读，非常适合快速开发。Python 编程简单直接，难度低于 Java，更适合初学者编程，让开发者更专注于解决问题本身而不是困惑于语言本身所带来的语法细节等。Python 几乎可以在各个领域使用，适用于各种平台，包括 Web 开发、网络运维、科学计算、3D 游戏和图形界面开发和人工智能等。

Python 之所以成为人工智能的首选实现语言，除了简单直接外，还因为它有优质的文档支持，丰富强大的 AI 库，海量的模块，这些都为人工智能提供了简单而强大的解决方案。

对于一个高中生而言，学好了 Python 将对他们进入大学后轻松驾驭各个领域的专业实验有很大的帮助。

13.10　总　　结

回顾本章中，我们学习了工智能定义及其发展，人工智能的种类；人工智能主要分为两个种类，一是运用符号思考的人工智能，二是运用神经网络思考的人工智能。

然后介绍了启发式算法、遗传算法和深度学习，并学习了什么是图灵测试，人工智能与机器人以及人工智能与 Python 的联系。

13.11　练　　习

（1）人工智能是什么？为什么要学习人工智能？

（2）图灵测试的原理是什么？

（3）人工智能技术越来越发达，人类会被取代吗？人类会成为人工智能的奴隶吗？怎样权衡人工智能与伦理问题的问题？

第 14 章　初识机器学习

本章将通过介绍机器学习的概念、机器学习的类型与机器学习相关算法等基本概念，理解什么是机器学习，通过本章的学习，需要掌握：

- 了解机器学习的基本概念。
- 理解机器学习中监督学习与无监督学习的区别。
- 理解 K-means 算法的工作原理。

14.1　机器学习的基本概念

2016 年 3 月，AlphaGo 与围棋世界冠军、职业九段棋手李世石进行围棋人机大战，以 4 比 1 的总比分获胜；2017 年 5 月，在中国乌镇围棋峰会上，它与排名世界第一的世界围棋冠军柯洁对战，以 3 比 0 的总比分获胜。围棋界公认阿尔法围棋的棋力已经超过人类职业围棋顶尖水平，随着 AlphaGo 的大火，机器学习（Machine Learning，ML）开始获得越来越多的关注。

那么什么是机器学习呢？美国卡内基梅隆大学机器学习研究领域的著名教授 Tom Mitchell（汤姆• 米歇尔）对机器学习的定义为：“A program can be said to learn from experience E with respect to some class of tasks T and performance measure P,if its performance at tasks in T,as measured by P,improves with experience E”，翻译过来就是：“如果一个程序在使用既有的经验（E）执行某类任务（T）的过程中被认定为是‘具备学习能力的’，那么它一定需要展现出利用现有的经验（E），不断改善其完成既定任务（T）的性能（P）的特质”。

机器学习是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。它专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能。它是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合而

不是演绎。

除去一些无关紧要的情况，人们很难直接从原始数据本身获得所需信息，例如，对于垃圾邮件的检测，侦测一个单词是否存在并没有太大的意义，然而当某几个特定单词同时出现时，再辅以考察邮件长度以及其他因素，人们就可以更准确地判定该邮件是否为垃圾邮件。简单地说，机器学习就是把无序的数据转换成有用的信息。

机器学习横跨计算机科学、工程技术和统计学等多个学科，需要多学科的专业知识。稍后你就能了解到，它也可以作为实际工具应用于从政治到地质学的多个领域，解决其中的很多问题。甚至可以这么说，机器学习对于任何需要解释并操作数据的领域都有所裨益。

机器学习的工作方式如下。

（1）选择数据：将数据分成训练数据、验证数据和测试数据三组。

（2）模型数据：使用训练数据来构建使用相关特征的模型。

（3）验证模型：使用验证数据接入模型。

（4）测试模型：使用测试数据检查被验证的模型的表现。

（5）使用模型：使用完全训练好的模型在新数据上做预测。

（6）调优模型：使用更多数据、不同的特征或调整过的参数来提升算法的性能表现。

14.2　机器学习的类型

机器学习的类型，从解决问题的方向来讲主要可以分为两大类：监督学习和非监督学习。第 13 章中我们提到的半监督学习也属于机器学习的类型，但限于篇幅，本章不做介绍，感兴趣的读者可以自己查找一些书籍学习。

14.2.1　监督学习

通过已有的训练样本（即已知数据以及其对应的输出）来训练，从而得到一个最优模型，再利用这个模型将所有新的数据样本映射为相应的输出结果，对输出结果进行简单的判断从而实现分类的目的，那么这个最优模型也就具有了对未知数据进行分类的能力。

我们在很小的时候就会被大人教授这是鸟，那是狗，这个是西瓜，那个是南瓜，这个可以吃，那个不能吃之类的，我们眼里见到的这些景物、食物就是机器学习中的输入，大

人告诉我们的结果就是输出，久而久之，当见得多了，大人说得多了，我们脑中就会形成一个抽象的模型，下次在没有大人提醒的时候看见别墅或者洋楼，我们也能辨别出来这是房子，不能吃，房子本身也不能飞等信息。上学的时候，老师教认字、数学公式、英语单词等，我们在下次碰到的时候，也能区分开并识别它们。这就是监督学习，它在我们生活中无处不在。在这里举个例子给大家简单地阐述一下监督学习的思想。

我们首先搜索一些橘子的图片如图 14.1 所示。



图 14.1　搜索图片

我们使用监督学习来找到所要的橘子的照片，而监督学习需要已标记的样本（包括特征值和对应的标签值）。学习样本和标签之间的对应关系，举一反三得到一个最优的模型，再利用这个模型将所有新的数据样本映射为相应的输出结果，如图 14.2 所示。

特征值1 → 标签值1

特征值2 → 标签值2

特征值3 → 标签值3

特征值4 → 标签值4

图 14.2　映射关系

对于上面搜索到的这些图片，有很多不是橘子的图片。例如，有很多带有黑色的图片，如图 14.3 所示左上角的图，它们其中一个特征是图片颜色以黑白为主，给这些图打上标签 0 表示这不是我们需要的图；而对于另一张图，特征就是图片颜色以橘子该有的颜色为主，于是打上标签 1 表示这是我们需要的图（没有标签值的称为非监督学习），如图 14.3 所示。

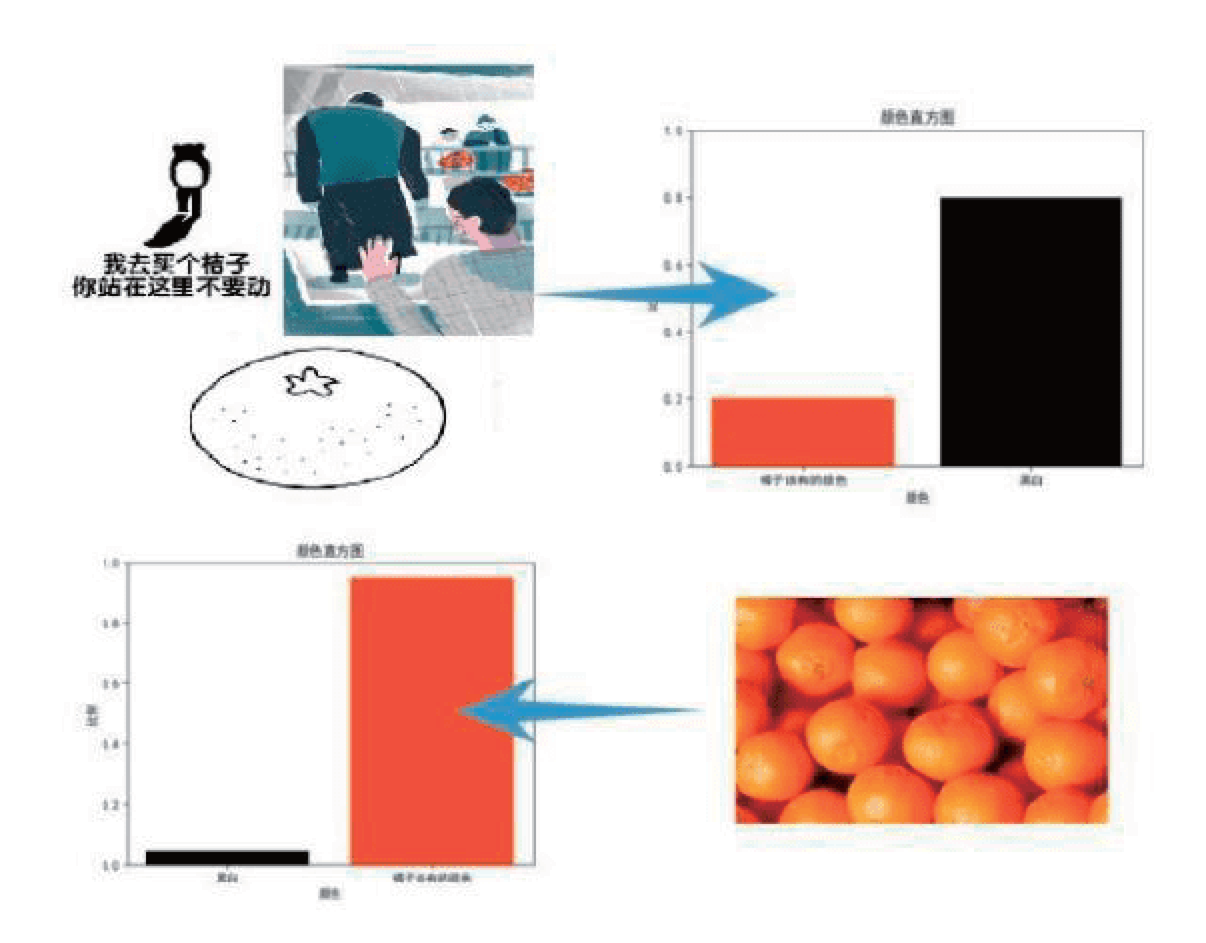


图 14.3　标签特征确定

由此通过训练我们就可以将图片分为两类，当收到一张没有标签的新图片时，就可以自动把它分到相应的类中，最后得出结果，如图 14.4 所示。

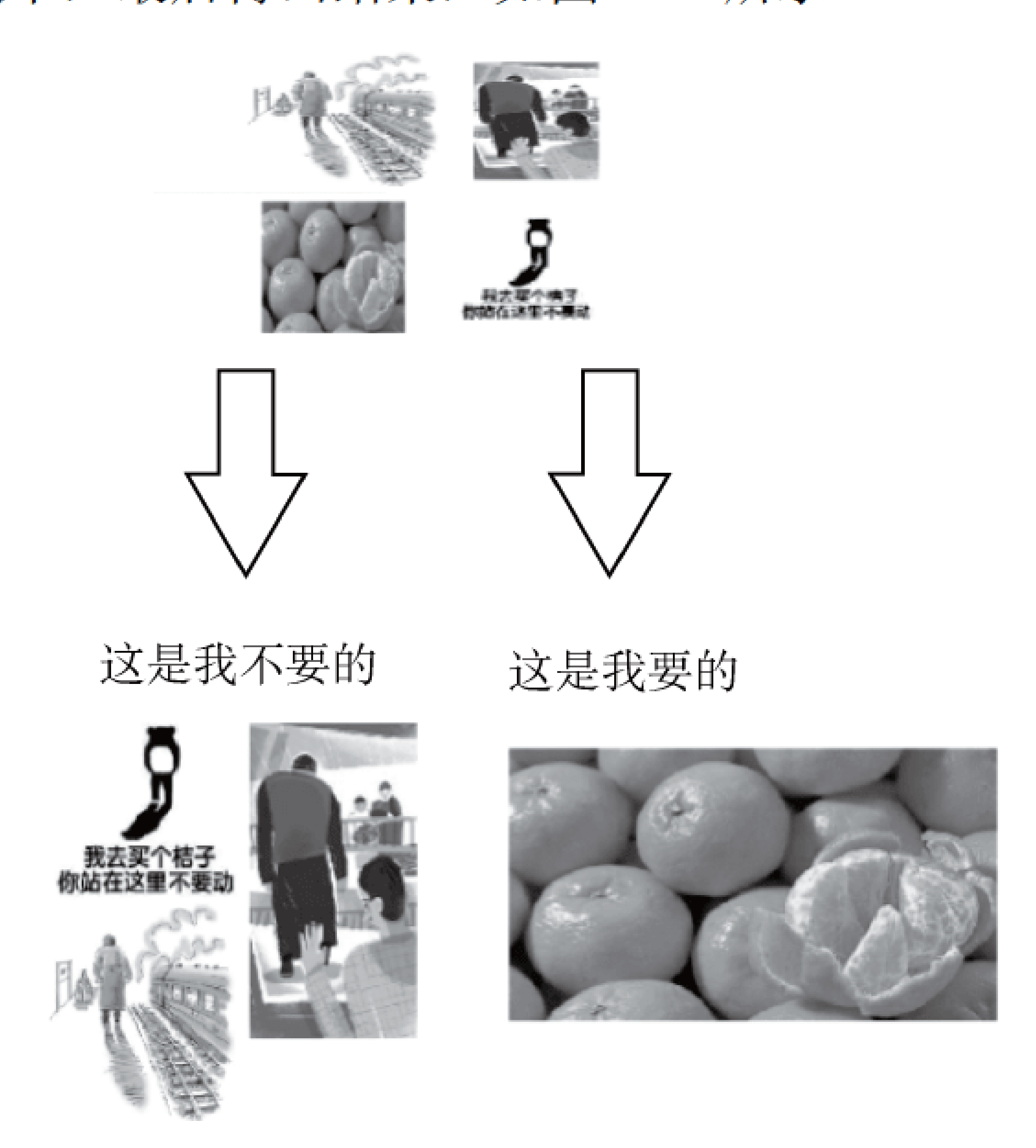


图 14.4　监督学习结果

在明白了监督学习的思想后，我们来学习监督学习中的两类非常重要的应用。

1. 分类

分类对于我们其实并不陌生，在现实生活中，经常会碰到这样的场景：“这个人是男的还是女的？”，我们通常会在大脑里考虑非常多的因素，如头发长短、身材曲线、穿衣风格等，在大脑中，我们会结合观察到的多种因素去判断男女，这就是一个分类问题。

分类的目的是学会一个分类函数或分类模型（也经常称作为分类器)，该模型能把数据库中的数据项映射到给定类别中的某一个类别。

分类属于一种有指导的学习，模型的学习是在被告知每一个训练样本属于哪个类的“指导”下进行的。并随机从样本群选取。每一个训练样本有一个特定的类标签与之对应，它不用于无指导的学习（聚类）。

主要的分类算法有 k 近邻和决策树。

（1）k 近邻。所谓的 k 近邻，就是依据最邻近的一个或者几个样本的类别，来决定待分类的样本所属的类别，步骤主要如下。

①计算训练样本和测试样本中每个样本点的距离（常见的距离度量有欧式距离、马氏距离等）。

②对上面所有的距离值进行排序。

③选前 k 个最小距离的样本。

④根据这 k 个样本的标签进行投票，得到最后的分类类别。

举个简单的例子，我们常说，物以类聚，人以群分，判别一个人是一个什么样品质特征的人，常常可以从他/她身边的朋友入手，所谓观其友，而识其人。我们判别图 14.5 中的圆点是属于哪一类数据时，可以从它的邻居下手。但一次性看多少个邻居呢？从图 14.5 中，你还能看到：

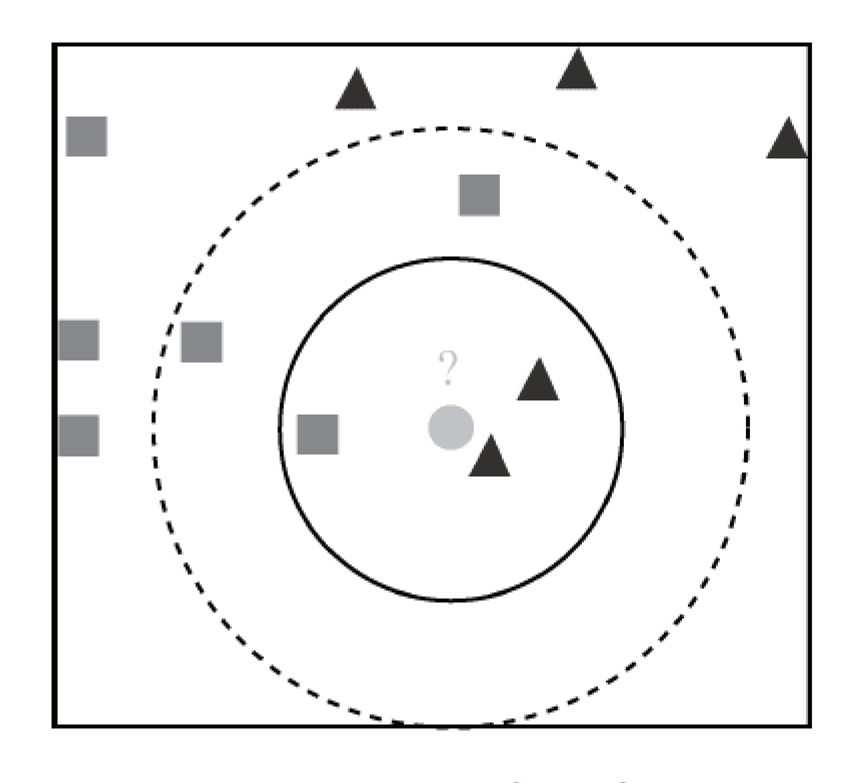


图 14.5　k 近邻距离

● 如果 k=3，圆点的最近的 3 个邻居是 2 个三角形和 1 个正方形，少数从属于多数，基于统计的方法，判定这个待分类点属于三角形一类。

● 如果 k=5，圆点的最近的 5 个邻居是 2 个三角形和 3 个正方形，还是少数从属于多数，基于统计的方法，判定这个待分类点属于正方形一类。

于此可以看到，当无法判定当前待分类点是从属于已知分类中的哪一类时，我们可以依据统计学的理论看它所处的位置特征，衡量它周围邻居的权重，而把它归为（或分配到）权重更大的那一类。这就是 k 近邻算法的核心思想。

（2）决策树。决策树（Decision Tree）是一个树结构（可以是二叉树或非二叉树）。其每个叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。

我们来用决策树执行分类操作看一下，程序全部代码请查阅配套的代码包，部分验证代码如程序 14.1 所示。

程序 14. 1　决策树算法：

```
import trees
myData,labels=trees.createDataSet()
print(labels)
myTree=treePlotter.retrieveTree(0)
print(myTree)
print(trees.classify(myTree,labels,[1,0]))
print(trees.classify(myTree,labels,[1,0]))
```

输出：

```
['no surfacing', 'flippers']
{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}
'no'
'yes'
```

分析：

根据上述输出结果。第一节点名为 no surfacing，它有两个子节点：一个是名字为 0 的

叶子节点，类标签为 no；另一个是名为 flippers 的判断节点，此处进入递归调用，flippers 节点有两个子节点。

2. 回归

回归问题预测的结果是连续的值，回归问题是要根据训练样本找到一个实值函数 $g(x)$。回归问题的要求是：给定一个新的模式，根据训练集推断它所对应的输出 y（实数）是多少。也就是使用 $y=g(x)$来推断任一输入 x 所对应的输出值。目前比较常见的回归模型如下。

（1）线性回归（Linear Regression）。在统计学中，线性回归是利用称为线性回归方程的最小平方函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。这种函数是一个或多个称为回归系数的模型参数的线性组合。只有一个自变量的情况称为简单回归，大于一个自变量的情况叫作多元回归。

（2）逻辑回归（Logistic Regression）。逻辑回归是用于处理因变量为分类变量的回归问题，常见的是二分类或二项分布问题，也可以处理多分类问题，它实际上是属于一种分类方法。二分类问题的概率与自变量之间的关系图形往往是一个 S 型曲线，采用 Sigmoid 函数实现。逻辑回归是一种广义的线性回归模型，除去 Sigmoid 映射函数关系，其他的步骤、算法都是线性回归的。可以说，逻辑回归，都是以线性回归为理论支持的。

14.2.2　无监督学习

无监督学习中使用的数据是没有标记过的，即不知道输入数据对应的输出结果是什么。无监督学习只能默默地读取数据，自己寻找数据的模型和规律，如聚类（把相似数据归为一组）和异常检测（寻找异常），无监督学习所回答的问题是“从这些数据中能发现什么？”

例如，我们去参观一个画展，虽然我们对艺术一无所知，但是欣赏完很多幅作品之后，在面对一幅新作品的时候，至少可以知道这幅作品是什么派别的，如更抽象一些还是更写实一点，尽管不能很清楚地领悟这幅画的含义，但是至少我们可以把它分为某一类。

举个例子，在无监督学习中给定的数据和监督学习中给定的数据是不一样的，在无监

督学习中给定的数据没有任何标签或者说只有同一种标签，如图 14.6 所示。

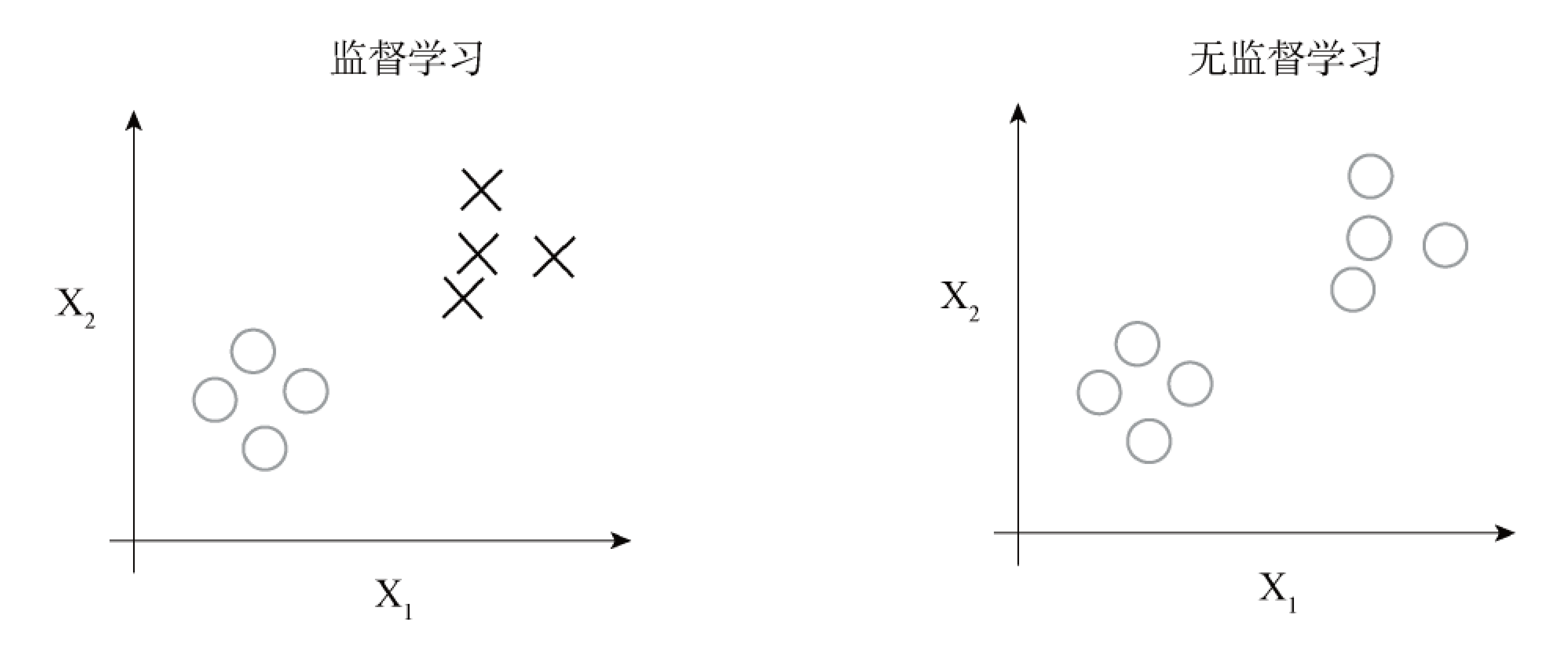


图 14.6　学习样本

在无监督学习中，我们基本上不知道结果会是什么样子，但可以通过聚类的方式从数据中提取一个特殊的结构，在给定了一组数据后，我们可以通过数据间的相似性发现这组数据中的特殊结构，进而将这组数据分成多个不同的簇，也就是它告诉了我们，这个数据能够这么分，但是并没有告诉我们，哪个是我们需要的，如图 14.7 所示，这就是无监督学习。

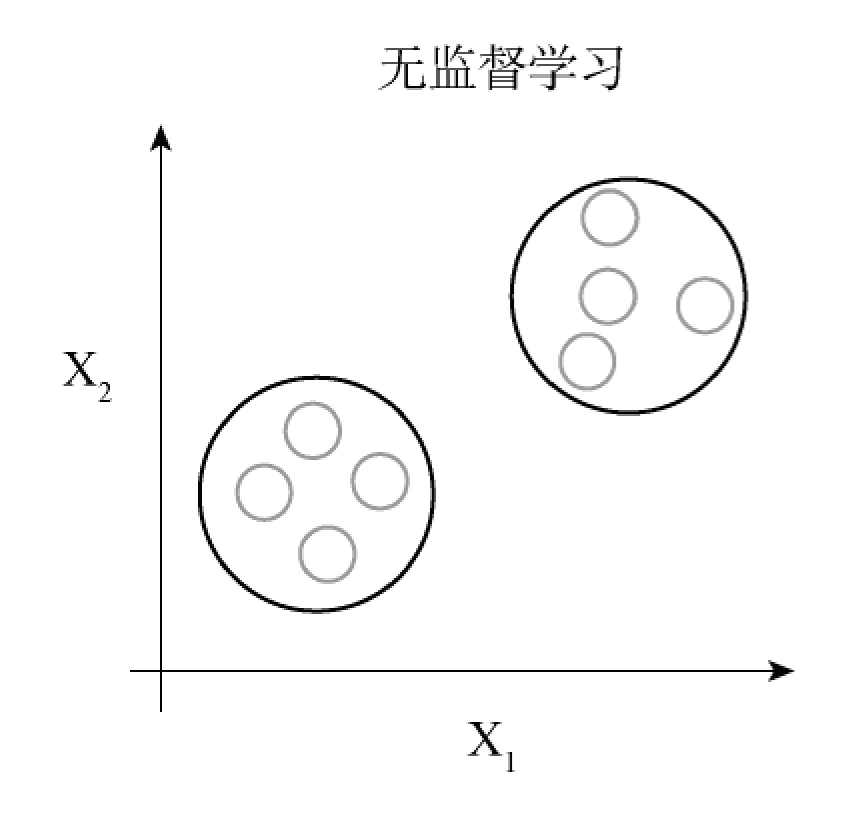


图 14.7　无监督学习结果

在无监督学习中，有两类重要的应用。

1. 数据聚类

数据聚类是无监督学习的主流应用之一，聚类就是对大量未知标注的数据集，按数据

的内在相似性将数据集划分为多个类别，使类别内的数据相似度较大而类别间的数据相似度较小。

给定一个有 n 个对象的数据集，构造数据的 k 个簇，$k \leqslant n$。满足下列条件：

（1）每一个簇至少包含一个对象。

（2）每一个对象属于且仅属于一个簇。

（3）将满足上述条件的 k 个簇称作一个合理划分。

基本思想：对于给定的类别数目 k，首先给出初始划分，通过迭代改变样本和簇的隶属关系，使得每一次改进之后的划分方案都较前一次好。

2. 特征降维

特征降维是无监督学习的另一个应用。当提取的数据特征维数太多时，经常会导致特征匹配时过于复杂，消耗系统资源，这时不得不采用特征降维的方法。所谓特征降维，即采用一个低维度的特征来表示高维度。我们会经常在实际项目中遭遇特征维度非常高的训练样本，而往往又无法借助自己的领域知识人工构建有效的特征；并且在数据表现方面，我们无法用肉眼观测超过三个维度的特征。因此，特征降维不仅重构了有效的低维度特征向量，同时也为数据展现提供了可能。

（1）主成分分析算法（Principal Component Analysis，PCA）。是最常用的线性降维方法，它的目标是通过某种线性投影，将高维的数据映射到低维的空间中表示，并期望在所投影的维度上数据的方差最大，以此使用较少的数据维度，同时保留住较多的原数据点的特性。

（2）局部线性嵌入（Locally Linear Embedding，LLE）。是一种非线性降维算法，它能够使降维后的数据较好地保持原有流形结构。LLE 可以说是流形学习方法最经典的工作之一。很多后续的流形学习、降维方法都与 LLE 有密切联系。

14.3　聚类案例：K-means 聚类算法

聚类是一种无监督学习的方法，即事先并不知道要寻找的目标，聚类将一堆没有标签，但特征相似的数据自动归类。下面将通过学生成绩评估的例子具体介绍聚类算法工作原理。某外语学校按照惯例在每个学期期末对学生半年的学习表现进行一次综合测评，其衡量指

标如表 14.1 所示。

表 14.1　英语成绩表

NO.	Spoken English	Written English
001	90	89
002	85	90
003	78	76
004	70	69
005	80	78
006	60	60
007	67	68

对应的成绩情况坐标图，如图 14.8 所示。

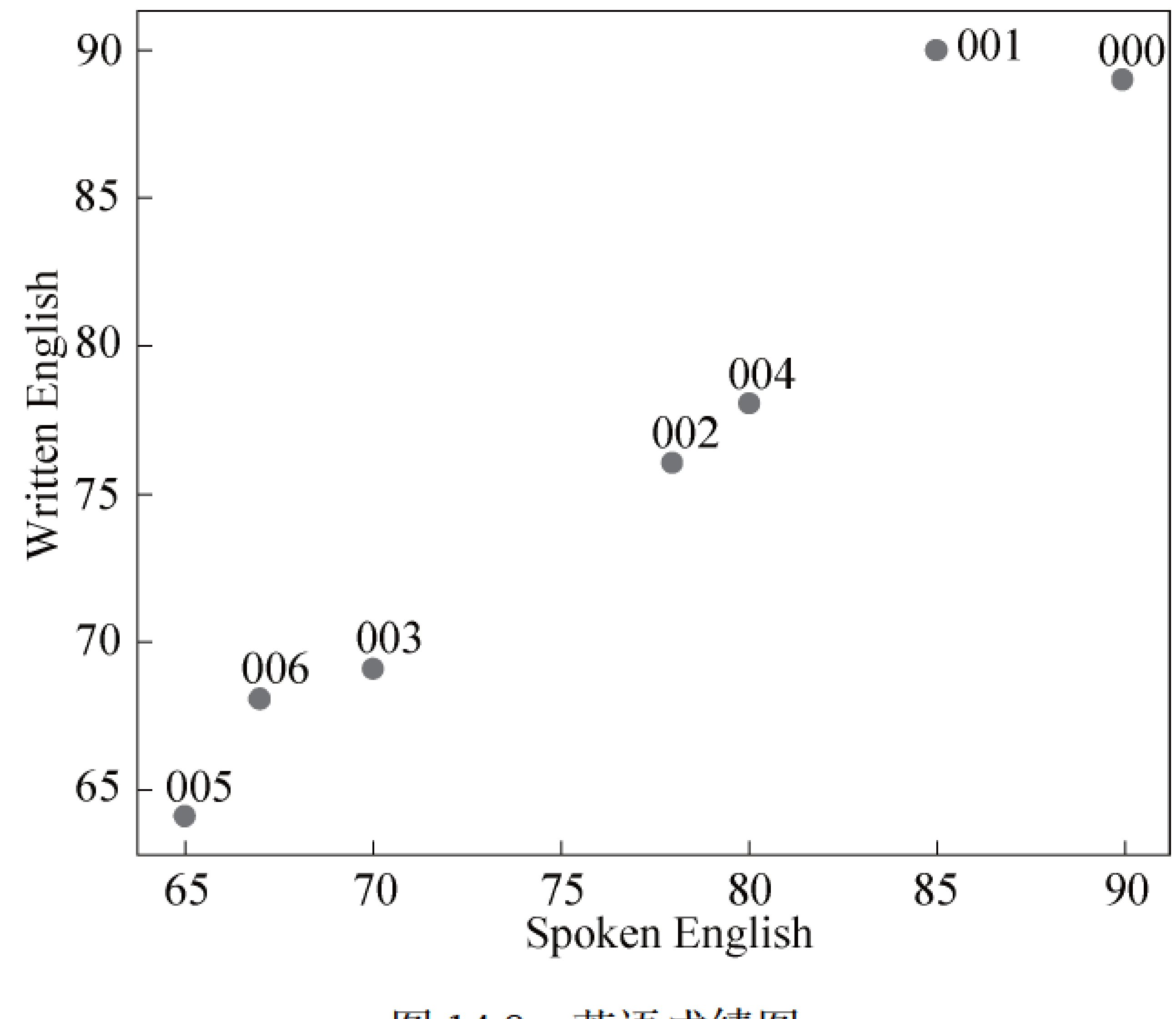


图 14.8　英语成绩图

为了避免试卷的难易程度对测评的影响，校方需要按照各指标的相似程度为学生评估 A、B、C 三个等级，现在的问题是应该采用何种方式划分成绩等级呢？显然，我们可以从案例中得到两个关键字：一是相似程度，一是三等级。对应于图 14.8，如果已经得知三等级的坐标，那么我们就可以基于两点间距离计算相似度，寻找图中每一点最近的等级坐标，将该点归为该等级，其结果如图 14.9 所示。

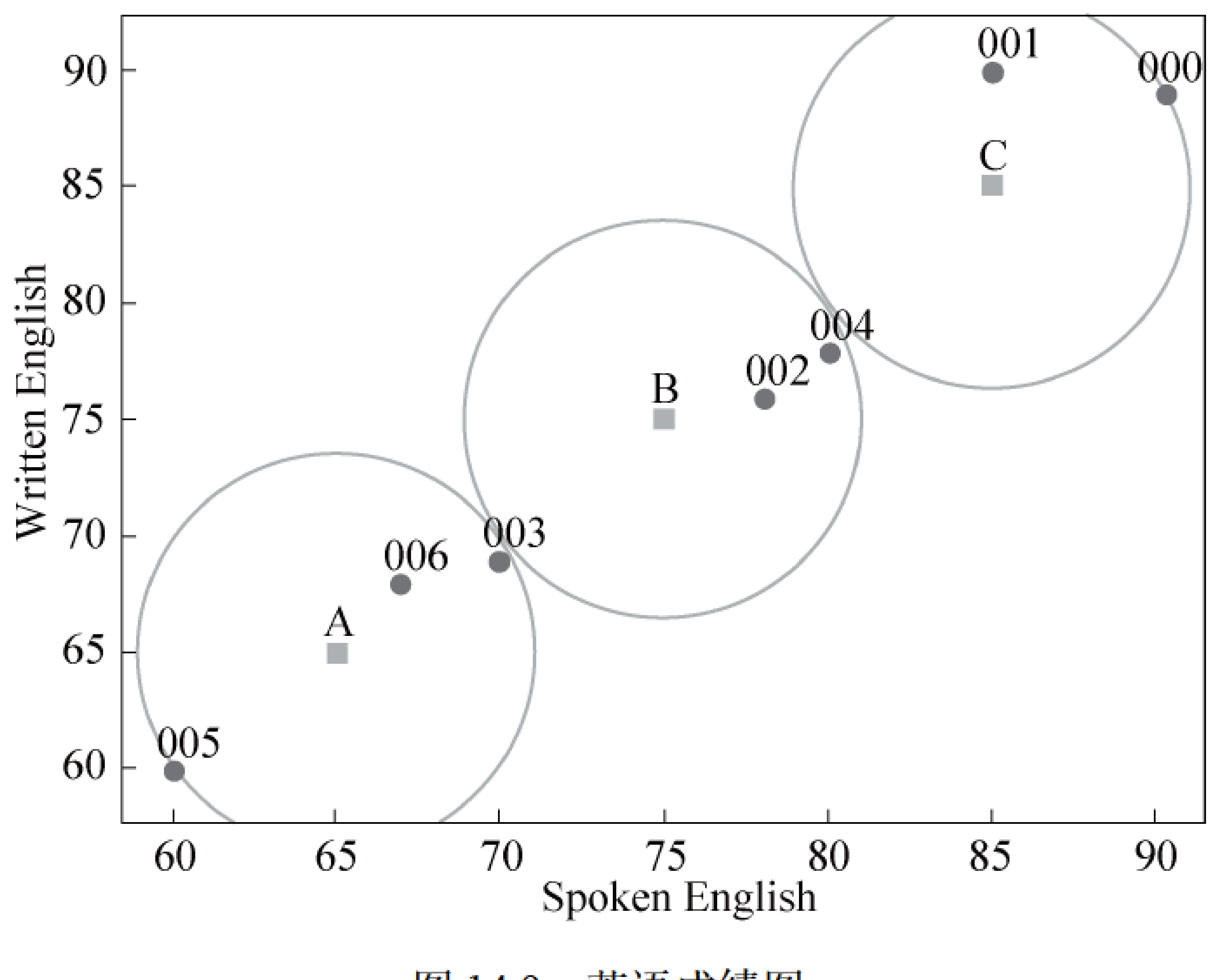


图 14.9　英语成绩图

这种根据对象与聚类质心之间的距离聚类对象的过程，属于无监督学习中常用的一种聚类方式，如 K-means 聚类。K-means 是最简单，但运用十分广泛的算法，其最终目标是把 n 个样本点划分到 k 个类簇中，使得每个点都属于离它最近的质心对应的类簇。现在让我们来看下 K-means 聚类算法的具体步骤：

（1）初始化质心。K-means 算法需要事先确定类簇分支数，并初始化各类簇的质心。

（2）聚类对象。K-means 算法按照对象与质心间的距离划分类簇，其中，距离可以是欧式距离 $d_{\text{Euclidean}}$：

$$d_{\text{Euclidean}} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{14-1}$$

或是余弦距离 d_{Cosine}：

$$d_{\text{Cosine}} = \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2} \bullet \sqrt{x_2^2 + y_2^2}} \tag{14-2}$$

（3）更新质心。K-means 完成对象聚类后，计算各类簇中对象的平均值，并以此作为新的质心。

现在结合表 14.1 的数据进一步整理 K-means 计算流程，首先随机初始化质心 $A = \{50,50\}$，$B = \{79,79\}$，$C = \{95,95\}$，接着以欧式距离聚类对象，其中属于类簇 A 的对

象有 {006}，而属于类簇 B、C 的对象分别为 {003,004,005,006}，{001,002}，在得到各类簇对象后，开始计算各类簇的平均值，其中 $mean_A = \{65,65\}$，$mean_B = \{73.75,72.75\}$，$mean_C = \{87.5,89.5\}$，最后更新质心，并重复上一步，直到所有簇的质心不再改变。接下来我们尝试着梳理算法的脉络，可构建出一个完整的 K-means 聚类流程，如图 14.10 所示。

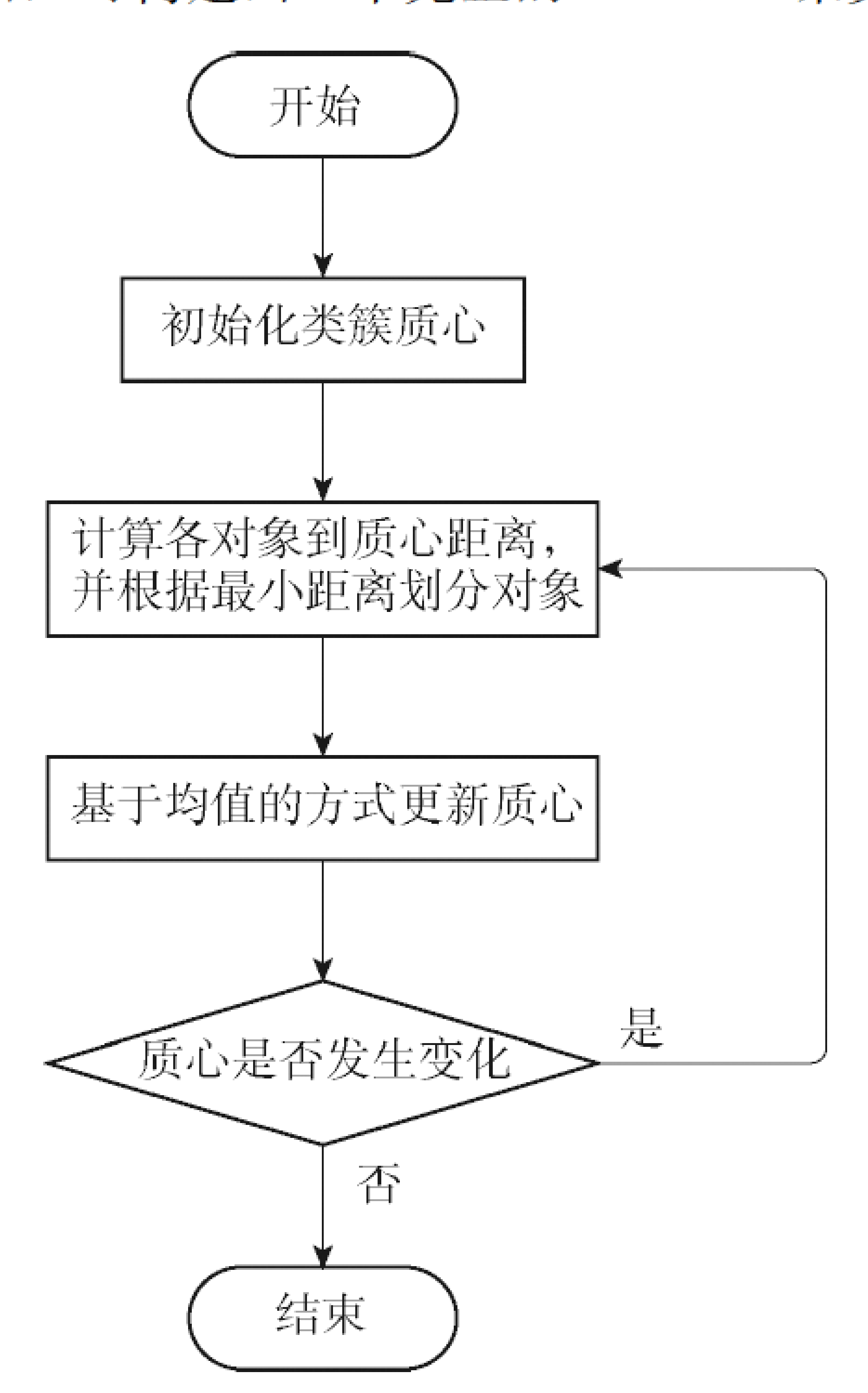


图 14.10　K-means 聚类算法流程图

再整理出 K-means 算法的一般步骤，编写代码如程序 14.2 所示。

程序 14.2　K-means 算法：

```
import numpy as np
from sklearn.cluster import KMeans
file_path = 'report.data'
dataset = np.loadtxt(file_path, delimiter=",")
kmeans = KMeans(init="random", n_clusters=3)
s = kmeans.fit(dataset)
print("the centroid of cluster are:\n{0}".format(kmeans.cluster_centers_))
print("the member of cluster are:{0}\n".format(kmeans.labels_))
```

输出：

```
the centroid of cluster are:
[[65.66666667 65.66666667]
 [79.          77.        ]
 [87.5         89.5       ]]
the member of cluster are:
[2 2 1 0 1 0 0]
```

分析：

首先我们通过 import 语句分别导入 numpy 模块和 scikit-learn 中的 KMeans 类，接着就可调用 numpy 模块的 loadtxt 方法加载表 14.1 中的数据，之后实例化 KMeans 对象，并调用其 fit 方法训练该对象，最后得到其训练后的 Kmeans 模型。值得注意的是，我们在实例化 KMeans 对象时，可选择其初始化类簇质心的方式，如 init="random"表示随机选择初始质心，也能够自定义其类簇数，这里的设置是 n_clusters=3，即最终该数据将聚类为 3 个类簇，K-means 算法是无监督学习的一个典型例子。

14.4　总　　结

我们在本章中了解了机器学习、机器学习的分类与有监督学习和无监督学习的基本概念，并学习了 Python 中相关模块的基本语法，最后通过 K-means 聚类算法的实际案例，深刻地理解了什么是机器学习，切实感受到了 Python 的模块化编程思想的强大与便捷。

14.5　练　　习

（1）经统计，2017 年全国各省消费数据如下：

北京	天津	河北	山西	福建	辽宁	吉林	上海	江苏	浙江
2959.1	2459.7	1495.6	1406.3	2709.5	1730.8	1561.8	3712.3	2207.5	2629.1

使用 K-means 算法对各省份消费水平进行分类（类别数为 2）。

（2）在某电影剧场中，上映的题材有爱情片、动作片等，如果一部电影中接吻镜头很

多，打斗镜头较少，显然是属于爱情片，反之为动作片。先对该剧场中电影题材的打斗动作和接吻动作数量进行评估，数据如下：

电影名称	打斗镜头	接吻镜头	电影类别
Califoria Man	3	104	爱情片
Beautigul Woman	1	81	爱情片
Kevin Longblade	101	10	动作片
Amped II	98	2	动作片

若给定一部电影数据(18,90)，表示打斗镜头为 18 个，接吻镜头为 90 个，请用 k 邻近算法判断该电影的类型。

第 15 章　自然语言处理

本章将学习什么是自然语言处理。我们将讨论一些处理文本的新概念，如分词（tokenization）、词干提取（stemming）、词形还原（lemmatization）等来处理文本。之后将讨论什么是词袋模型（Bag of Words）并如何使用它来对文本进行分类。我们将会弄明白如何使用机器学习来分析给定句子的意思。最后将讨论主题建模（topic modeling）。学完本章后，将会了解：

- 什么是自然语言处理。
- 文本分词。
- 使用 stemming 还原词汇。
- 使用 lemmatization 还原词汇。
- 文本分块。
- 使用词袋模型提取文章中的词频矩阵。
- 构建性别识别器。

15.1　什么是自然语言处理

自然语言处理（Natural Language Processing，NLP）（见图 15.1）是计算机科学领域与人工智能领域中的一个重要方向。它研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法。自然语言处理是一门融语言学、计算机科学、数学于一体的科学。因此，这一领域的研究将涉及自然语言，即人们日常使用的语言，所以它与语言学的研究有着密切的联系，但又有重要的区别。自然语言处理并不是一般地研究自然语言，而在于研制能有效地实现自然语言通信的计算机系统，特别是其中的软件系统。因而它是计算机科学的一部分。

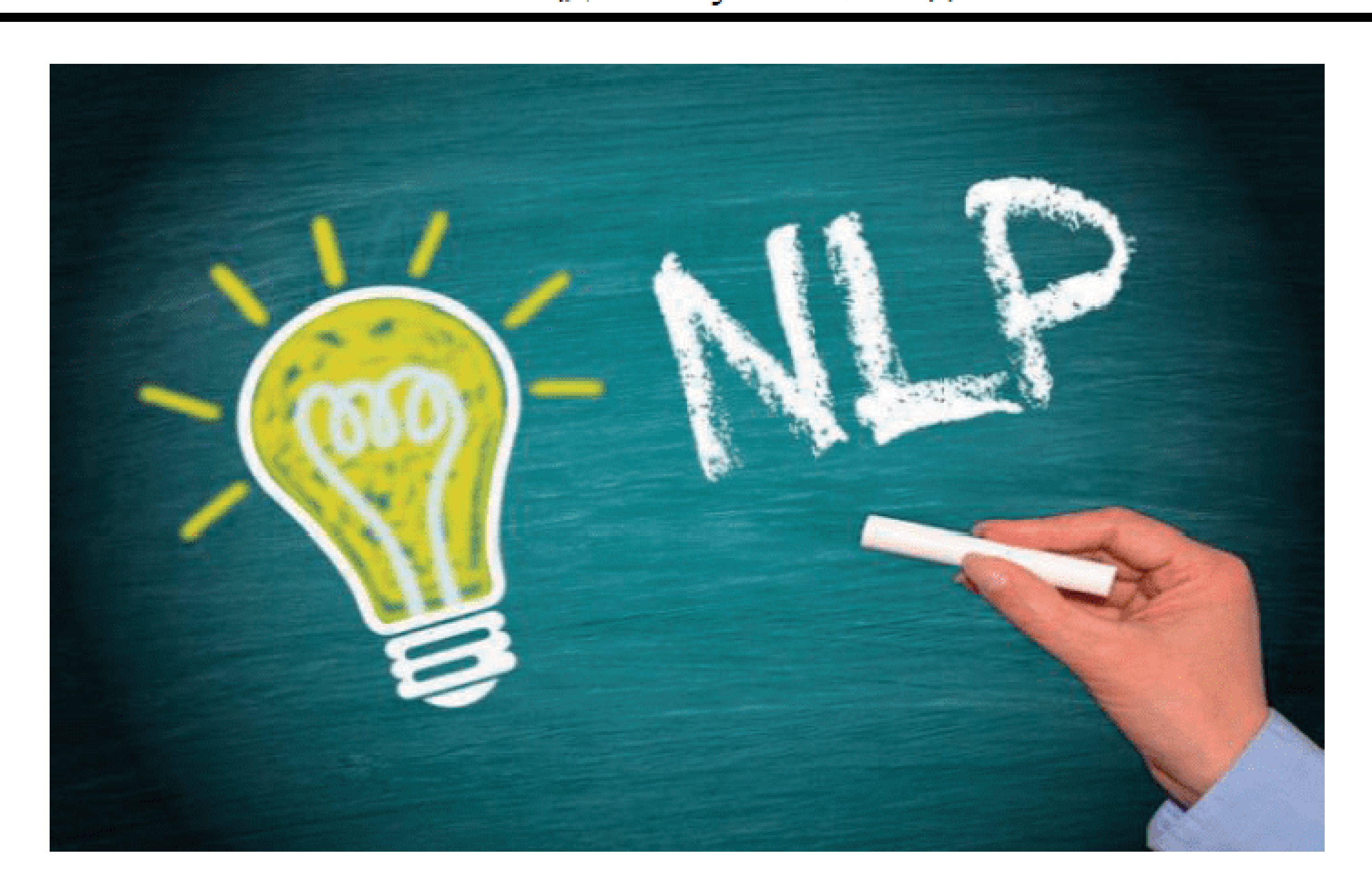


图 15.1　自然语言处理

自然语言处理（NLP）已经成为现代系统的重要组成部分。它广泛应用于搜索引擎、人机对话接口、文档处理等，如图 15.2 所示。机器可以很好地处理结构化数据。但是，当涉及无固定形式的文本时，它将很难处理。NLP 的目标是开发一种算法，使计算机能够理解无结构的文本，并帮助它们理解这种语言。



图 15.2　自然语言处理的应用

处理无结构自然语言的最大的一个挑战是词的数量多，变化大。上下文在如何理解特定的句子方面起着非常重要的作用。人类在这方面很擅长，因为我们为此已经训练了很多年了。我们可以马上使用我们过去的知识去理解上下文并知道对方在说什么。

针对这个问题，NLP 研究人员开始使用机器学习方法开发各种应用程序。要构建这样的应用程序，需要收集大量的文本，然后训练算法执行各种任务，如对文本进行分类、分析语义或主题建模。这些算法被训练去捕获输入文本数据以及其衍生的意思。

在这一章中，我们将讨论用于分析文本和构建 NLP 应用程序的各种基础概念，这将使我们能够理解如何从给定的文本数据中提取有意义的信息。在本章的开始，我们需要先导入 Python 的第三方库 NLTK（Natural Language Toolkit），它是一个自然语言处理工具包，在之后的程序中我们会用到它。此外，为了能够访问 NLTK 提供的数据集，我们需要下载这些数据集。我们需要在命令提示符下进入 Python 环境，并输入以下代码来下载这些数据集：

```
>>> import nltk
>>> nltk.download()
```

这样在之后的编程中，我们就能够使用 NLTK 提供的数据集了。另外，我们还将在本章中使用一个名为 gensim 的 Python 包，它是一个健壮的语义模型库，对于很多应用是很有帮助的，在之后的编程中我们会详细地介绍 gensim 包。为了让 gensim 更好地发挥作用，还需要安装一个名为 pattern 的软件包。在进行编程之前，请确保安装了 NLTK 和 gensim。

提示：

（1）在 Python 2.X 的版本中，如果只安装 gensim 包，程序有可能会报错，会提示缺少 pattern 包，所以还需要安装 pattern 包。

（2）在 Python 3.X 的版本中，pattern 包进行了升级，如果要安装它，请安装 pattern3，它是 pattern 包的升级。

15.2　文 本 分 词

当处理文本时，我们需要将它拆分成小片来进行分析。它是将输入的文本分成像单词或句子的一小片，这些片被称之为原型。我们可以根据自己所想，定义自己的方法将文本分成许多片。

例如，用这样一个句子来进行分词：Do you know what natural language processing is? This is a very interesting technology! We'll look at it in this section.

如果我们是以句子为原型进行分词，那么结果是这样的：

'Do you know what natural language processing is?' ' This is a very interesting technology!'

'We’ll look at it in this section.'

如果我们以单词为原型进行分词，那么结果是这样的：

'Do', 'you', 'know', 'what', 'natural', 'language', 'processing', 'is', '?', 'This', 'is', 'a', 'very', 'interesting', 'technology', '!', 'We', "'ll", 'look', 'at', 'it', 'in', 'this', 'section', '.'

如果我们按照标点符号来进行划分的话，那么它们也会被单独划分出来。

下面用一段程序来实现文本的分词，如程序 15.1 所示。

程序 15.1　文本分词：

```
from nltk.tokenize import sent_tokenize, word_tokenize, WordPunctTokenizer

input_text = "Do you know what natural language processing is? This is a very
interesting technology! We'll look at it in this section."

print("\nSentence tokenizer:")
print(sent_tokenize(input_text))

print("\nWord tokenizer:")
print(word_tokenize(input_text))

print("\nWord punct tokenizer:")
print(WordPunctTokenizer().tokenize(input_text))
```

输出：

```
 Sentence tokenizer:
['Do you know what natural language processing is?', 'This is a very interesting technology!', "We'll look at
it in this section."]

Word tokenizer:
['Do', 'you', 'know', 'what', 'natural', 'language', 'processing', 'is', '?', 'This', 'is', 'a', 'very', 'interesting',
'technology', '!', 'We', "'ll", 'look', 'at', 'it', 'in', 'this', 'section', '.']

Word punct tokenizer:
['Do', 'you', 'know', 'what', 'natural', 'language', 'processing', 'is', '?', 'This', 'is', 'a', 'very', 'interesting',
'technology', '!', 'We', "'", 'll', 'look', 'at', 'it', 'in', 'this', 'section', '.']
```

分析：

本程序中我们先导入 NLTK 程序包，用于导入不同形式的分词器。然后定义了将用于分词

的输入文本。之后我们分别使用不同的分词器对输入文本进行分词。sent_tokenize 是将输入文本以句子为原型进行划分的；word_tokenize 是将输入文本以单词为原型进行划分的，此外，它还包括每一句最后的标点符号；WordPunctTokenizer 与 word_tokenize 的不同之处是句子当中的标点也会被划分出来。例如，We'll 中的“'”也被划分出来了。

提示：

NLTK（Natural Language Toolkit）是一套基于 Python 的自然语言处理工具集。它用来处理人类自然语言数据，提供了易于使用的接口，通过这些接口可以访问超过 50 个语料库和词汇资源（如 WordNet），还有一套用于分类、标记化、词干标记、解析和语义推理的文本处理库等。

中文分词的情况就比较麻烦了。在中文文本中，所有的词语连接在一起，计算机并不知道一个字应该与其前后的字连成词语，还是应该自己形成一个词语。因此我们对文本构建词袋之前，需要先借助额外的手段将文本中的词语分隔开。中文分词方法大多基于匹配与统计学方法，我们在这里不做详细介绍了。

15.3　使用 stemming 还原词汇

对于一些变化的词汇，我们必须处理相同单词的不同形式，并且使计算机能够明白这些不同的单词有相对的形式。例如，单词 write 能够以很多形式出现，如 wrote、writer、writing 等，如图 15.3 所示。我们刚刚看到的是有相同意思的一系列的单词。人类能够轻松地识别这些单词的基础形式和衍生形式。

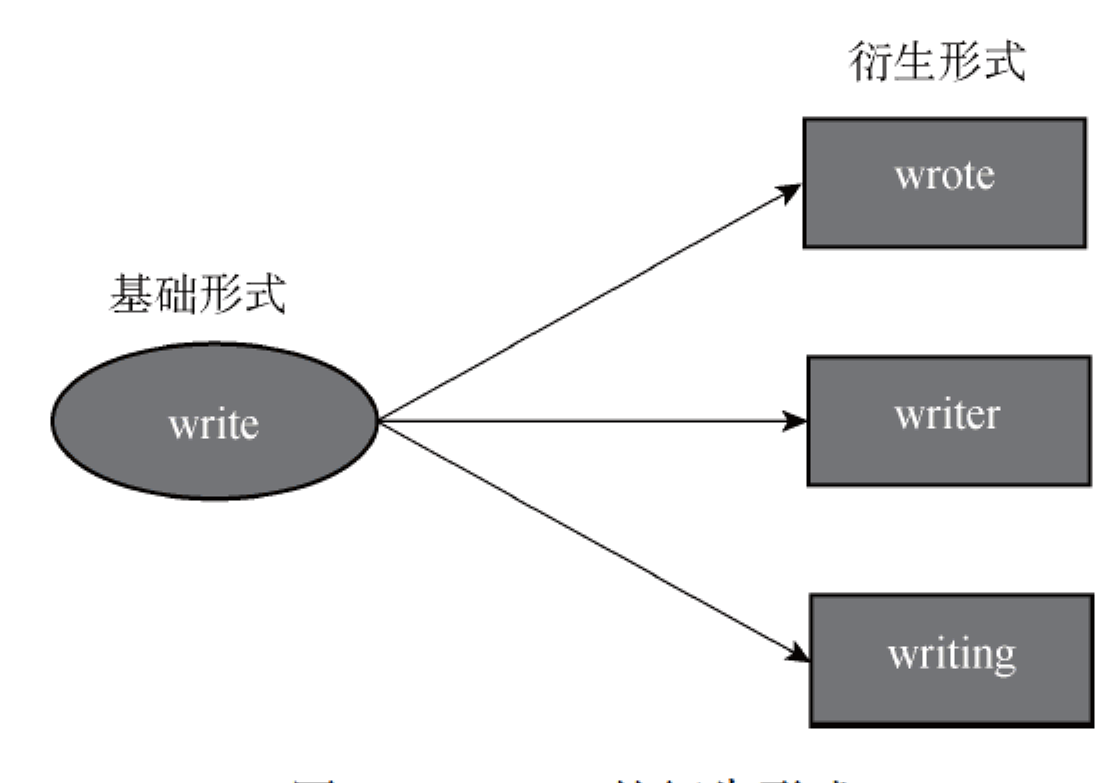


图 15.3　write 的衍生形式

当我们分析文本时，提取这些基本形式是很有用的。它将使我们能够提取有用的统计信息来分析输入文本。词干提取（stemming）能够做到这一点。词干提取器（stemmer）的目的是通过将单词的不同形式转换为基本形式来减少单词量。去掉单词的尾部将其变成基本形式是一个启发式的过程。

提示：

启发式指人在解决问题时所采取的一种根据经验规则进行发现的方法。其特点是在解决问题时，利用过去的经验，选择已经行之有效的方法，而不是系统地、以确定的步骤去寻求答案。

下面将用一个例子带你更加详细地了解 stemming 的原理。我们用'reading', 'calves', 'acting', 'undefined', 'house', 'possibly', 'version', 'hotel', 'learned', 'experience'这些单词作为测试，来看看 stemming 是如何工作的，如程序 15.2 所示。

程序 15.2　使用 stemming 还原词汇：

```
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.snowball import SnowballStemmer

input_words = ['reading', 'calves', 'acting', 'undefined', 'house',
                'possibly', 'version', 'hotel', 'learned', 'experience']

porter = PorterStemmer()
lancaster = LancasterStemmer()
snowball = SnowballStemmer('english')

stemmer_names = ['Porter', 'Lancaster', 'Snowball']
formatted_text = '{:>16}' * (len(stemmer_names) + 1)
print('\n', formatted_text.format('Input Word', *stemmer_names), '\n', '*'*70)

for word in input_words:
    output = [word, porter.stem(word),
              lancaster.stem(word), snowball.stem(word)]
    print(formatted_text.format(*output))
```

输出：

```
   Input Word          Porter       Lancaster        Snowball
**********************************************************************
      reading            read            read            read
```

```
    calves         calv        calv          calv
    acting          act         act           act
 undefined       undefin     undefin       undefin
     house          hous        hous          hous
   possibly      possibl        poss       possibl
   version       version        vert       version
     hotel         hotel       hotel         hotel
   learned         learn       learn         learn
experience        experi      expery        experi
```

分析：

在这个程序中，我们先导入了 3 个不同的词干提取器，它们分别是 porter、lancaster、snowball。然后定义了一些单词作为输入来进行测试。在程序的第 8～10 行我们创建了 3 个提取器对象。第 12～14 行我们创建了一个显示表格并格式化了输出文本。第 16～19 行遍历输入单词并使用 3 个不同的词干提取器提取词根。

我们讨论一下这里使用的 3 种词干提取算法。它们基本上都是为了实现相同的目标。它们之间的区别在于还原成基本形式的严格程度是不同的。

在严格程度上，porter 词干提取器是最不严格的，而 lancaster 是最严格的。如果仔细观察输出，你会注意到它们之间的区别。当涉及单词如 possibly 或 version 时词干提取器的行为会有所不同。从 lancaster 词干提取器获得的输出有点模糊，因为它减少了很多尾词。而且，这个算法非常快。一个很好的经验是使用 snowball 词干提取器，因为它在速度和严格程度之间有很好的平衡。

15.4　基于词义的词形还原

lemmatization 是另一种词形还原的方式。在前一节中，我们看到从词干中提取词的基本形式有时候可能没有任何意义。例如，3 个词干提取器都显示 calves 的基本形式是 calv，但它并不是一个真正的单词。lemmatization 采取了一种更具结构化的方法解决了这个问题。

lemmatization 原理是使用语法和词态分析器进行单词分析。它包含去除了如 ing 和 ed 等后缀的单词的基本形式。所有基本形式的单词集合被称作字典。如果使用 lemmatization 对 calves 进行词形还原，将输出 calf。值得注意的是单词基本形式的输出依赖于该词是动

词还是名词。

这里，我们将使用 lemmatization 来对上一节例子中的单词进行词形还原，进而比较它们之间的区别。我们分别使用名词还原器和动词还原器对单词进行词形还原，如程序 15.3 所示。

程序 15.3　使用 lemmatization 还原词汇：

```
from nltk.stem import WordNetLemmatizer

input_words = ['reading', 'calves', 'acting', 'undefined', 'house',
                'possibly', 'version', 'hotel', 'learned', 'experience']

lemmatizer = WordNetLemmatizer()

lemmatizer_names = ['Noun Lemmatizer', 'Verb Lemmatizer']
formatted_text = '{:>24}' * (len(lemmatizer_names) + 1)
print('\n', formatted_text.format('Input Word', *lemmatizer_names), '\n', '*'*74)

for word in input_words:
    output = [word, lemmatizer.lemmatize(word, pos='n'),
              lemmatizer.lemmatize(word, pos='v')]
    print(formatted_text.format(*output))
```

输出：

```
Input Word          Noun Lemmatizer          Verb Lemmatizer
**************************************************************************
  reading              reading                  read
   calves               calf                    calve
   acting              acting                   act
  undefined           undefined               undefined
   house               house                   house
  possibly            possibly                possibly
  version             version                 version
   hotel               hotel                   hotel
  learned             learned                  learn
  experience          experience              experience
```

分析：

我们先导入了一个词形还原器 WordNetLemmatizer。接着我们还是使用上节中使用的

输入单词来进行测试，以便比较它们的输出有什么不同。程序的第 8～10 行是创建显示列表并格式化文本。程序的第 12～15 行遍历输入单词并分别使用动词还原器和名词还原器来还原词汇。

我们可以看到，当遇到形如 reading 或者 calves 这些单词时，名词还原器和动词还原器的还原结果是不一样的。如果将这些输出与之前的词干提取器的输出结果相比，这两者的结果也有不同。lemmatizer 输出都是有意义的，而 stemmer 输出可能有意义也可能没有意义。

15.5　文本分块

文本数据经常需要被分成几小块来进行分析，这个过程称之为分块。这种技术在文本分析中使用频繁。使用文本分块的情况变化很多，各不相同，这依赖于手头的项目。文本分块与分词不同。在分块时，我们不受任何条件的限制，并且输出的结果是有意义的。

当我们处理大篇幅的文本文档时，将文本分块就显得很重要，这有利于提取有意义的信息。我们可以根据单词的数量来进行分块，也可以根据其他的一些条件来分块。下面我们来实现程序 15.4，用于将输入的文本进行分块。

程序 15.4　文本分块：

```
import numpy as np
from nltk.corpus import brown

def chunker(input_data, N):
    input_words = input_data.split(' ')
    output = []
    cur_chunk = []
    count = 0
    for word in input_words:
        cur_chunk.append(word)
        count += 1
        if count == N:
            output.append(' '.join(cur_chunk))
            count, cur_chunk = 0, []
    output.append(' '.join(cur_chunk))
    return output

```

```
18:  if __name__=='__main__':
19:      input_data = ' '.join(brown.words()[:6300])
20:      chunk_size = 800
21:      chunks = chunker(input_data, chunk_size)
22:      print('\nNumber of text chunks =', len(chunks), '\n')
23:      for i, chunk in enumerate(chunks):
24:          print('Chunk', i + 1, '==>', chunk[:50])
```

输出：

```
Number of text chunks = 8
Chunk 1 ==> The Fulton County Grand Jury said Friday an invest
Chunk 2 ==> 2 , 1913 . They have a son , William Berry Jr. , a
Chunk 3 ==> bonds . Schley County Rep. B. D. Pelham will offer
Chunk 4 ==> Texas was a republic " . It permits the state to
Chunk 5 ==> the requirement that each return be notarized . In
Chunk 6 ==> the Education courses . Fifty-three of the 150 rep
Chunk 7 ==> drafts of portions of the address with the help of
Chunk 8 ==> plan alone would boost the base to $5,000 a year a
```

分析：

这个程序的主要作用是将文本进行分块。首先，我们先导入 numpy 库（Python 的一个扩充程序库，支持高级大量的维度数组与矩阵运算）。接着我们从 NLTK 的语料库中导入 Brown 语料库，它作为我们将要进行分块的文本。接着我们定义一个分块函数，这个函数接受两个参数，第一个参数是输入文本，第二个参数是每一块的单词数量。在该函数中我们遍历输入文本中的单词并使用输入参数将它们分块，函数返回一个列表。随后我们定义主函数，并且读入 Brown 语料库中的文本。我们读入了文本中前 6300 个单词，读取多少单词可以自己决定。程序的第 20 行定义了每块的单词数。程序的第 21～24 行是将输入文本分块并显示输出结果。

提示：

Brown 语料库（Brown Corpus）是 20 世纪 60 年代初美国 Brown 大学创建的。Brown 语料库收集了 500 个连贯英语书面语，每个文本超过 2000 词，整个语料库约 1014300 词。该语料库是第一个机读语料库，也是第一个平衡语料库。它用于研究当代美国英语。尽管由现在理论及技术的水准看来，Brown 的资料及平衡方式略显粗糙，可是这个语料库一直是（英语）平衡语料库的标准。

程序 15.4 一共输出了 8 个块，每个块显示了前 50 个字符。

15.6　使用词袋模型提取词频矩阵

Bag of Words，也称作“词袋”。它是用于描述文本的一个简单数学模型，也是常用的一种文本特征提取方式。在信息检索中，词袋模型假定对于一个文本，忽略其次序和语法，将其仅仅看作是若干词汇的集合。文档中每个单词的出现都是独立的，不依赖于其他单词是否出现。也就是说，文档中任意一个位置出现的任何单词，都不受该文档语意影响而是独立选择的。

文本分析的主要目标之一是将文本转换成数值形式，这样就可以在上面使用机器学习了。让我们考虑一下包含数百万个单词的文本文档。为了分析这些文档，我们需要提取文档，并将其转换为数值形式。

机器学习算法需要处理数值形式的数据，以便它们能够分析数据并且提取有用的信息。我们可以用词袋模型从文档的所有单词中提取特征单词，并且用这些特征项矩阵建模。这就使得我们能够将每一份文档描述成一个词袋。我们只需要记录单词的数量，语法和单词的顺序都可以忽略。

那么一份文档的单词矩阵是怎样的呢？一个文档的单词矩阵是一个记录出现在文档中的所有单词的次数。因此，一份文档能被描述成各种单词权重的组合体。我们能够设置条件，筛选出更有意义的单词。而且，我们能构建出现在文档中所有单词的频率直方图，这就是一个特征向量。这个特征向量被用于文本分类。

思考下面的句子。

句 1：The children are playing in the playground.

句 2：The playground has a lot of space.

句 3：Lots of children like playing in an open space.

通过思考上面的三个句子，我们能够得到下面 14 个唯一的单词：

the　children　are　playing　in　playground　has　a　lot　of　space　like　an　open

这里有 14 个不同的单词。我们可以用出现在每句话中的单词次数为每一句话构建一个直方图。每一个特征向量都将有 14 维，因为有 14 个不同的单词：

句 1：[2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]

句 2：[1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0]

句 3：[0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1]

既然已经提取了这些特征向量，我们就可以使用机器学习算法来分析这些数据。编写程序 15.5 来使用词袋模型提取一个词频矩阵。

程序 15.5　使用词袋模型提取词频矩阵：

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import brown
from text_chunker import chunker

input_data = ' '.join(brown.words()[:5600])
chunk_size = 900
text_chunks = chunker(input_data, chunk_size)

chunks = []
for count, chunk in enumerate(text_chunks):
    d = {'index': count, 'text': chunk}
    chunks.append(d)

count_vectorizer = CountVectorizer(min_df=7, max_df=18)
document_term_matrix = count_vectorizer.fit_transform([chunk['text'] for
                                                        chunk in chunks])

vocabulary = np.array(count_vectorizer.get_feature_names())

chunk_names = []
for i in range(len(text_chunks)):
    chunk_names.append('Chunk ' + str(i + 1))

print("\nDocument Term Matrix:")
formatted_text = '{:>9}' * (len(chunk_names) + 1)
print('\n', formatted_text.format('Word', *chunk_names), '\n', '*'*76)
for word, item in zip(vocabulary, document_term_matrix.T):
    output = [word] + [str(freq) for freq in item.data]
    print(formatted_text.format(*output))
```

输出：

Document Term Matrix:

Word	Chunk 1	Chunk 2	Chunk 3	Chunk 4	Chunk 5	Chunk 6	Chunk 7
and	27	5	15	8	16	17	6
are	2	2	1	1	4	1	1
as	6	4	4	2	9	3	3
be	6	11	5	10	2	3	2
by	3	5	4	10	10	7	3
for	9	12	5	13	7	5	2
his	4	5	5	2	2	5	1
in	15	16	12	14	19	20	3
is	3	7	5	1	7	5	1
it	9	6	12	6	1	3	2
of	33	24	28	36	37	33	3
on	4	6	3	15	1	6	4
one	1	3	1	4	1	1	1
or	2	2	1	1	2	1	1
the	77	57	48	58	48	72	9
to	11	34	22	26	18	19	5
was	5	8	9	5	6	6	1
with	2	2	4	1	3	3	1

（表头下方原有一行星号分隔线 **）

分析：

在本程序中我们使用词袋模型提取一个词频矩阵。我们从 sklearn（用于数据挖掘和机器学习等领域，封装了大量的机器学习算法）的特征提取模块中导入 CountVectorizer，它可以把收集到的文本文档数据转换成单词矩阵，并对单词进行计数。我们还是使用 Brown 语料库作为输入文本，将上一节中的分块函数导入进来。这次，我们从 Brown 语料库中读入 5600 个单词，并定义每块的单词数为 900，然后使用 chunker 函数将文本进行分块。程序的第 10～13 行将所分的块转换为字典项。接着我们使用 CountVectorizer 方法对单词进行计数，并提取单词矩阵。该方法需要两个输入参数，第一个参数是出现在文档中单词的最小频率度，第二个参数是出现在文档中的单词的最大频率度。这两个频度参考文本中单词的出现次数。

max_df：可以设置为范围在[0.0 1.0]的 float，也可以设置为没有范围限制的 int，默认

为 1.0。这个参数的作用是作为一个阈值，当构造语料库的关键词集的时候，如果某个词的词频大于 max_df，这个词不会被当作关键词。如果这个参数是 float，则表示词出现的次数与语料库文档数的百分比，如果是 int，则表示词出现的次数。如果参数中已经给定了词汇，则这个参数无效。

min_df：类似于 max_df，不同之处在于如果某个词的词频小于 min_df，则这个词不会被当作关键词。

程序的第 19 行是提取词汇表，它指的是在前一步中提取的不同单词的列表。接着我们为每一个块生成显示名称。最后，打印该单词矩阵。在程序的输出中，我们能够看到文档的单词矩阵和每个单词在每一块的出现次数。

词袋模型非常简单，但还需要与一些文本处理技术相搭配才能在应用中取得较好的效果。图 15.4 展示了利用词袋模型构建文本特征的基本流程。

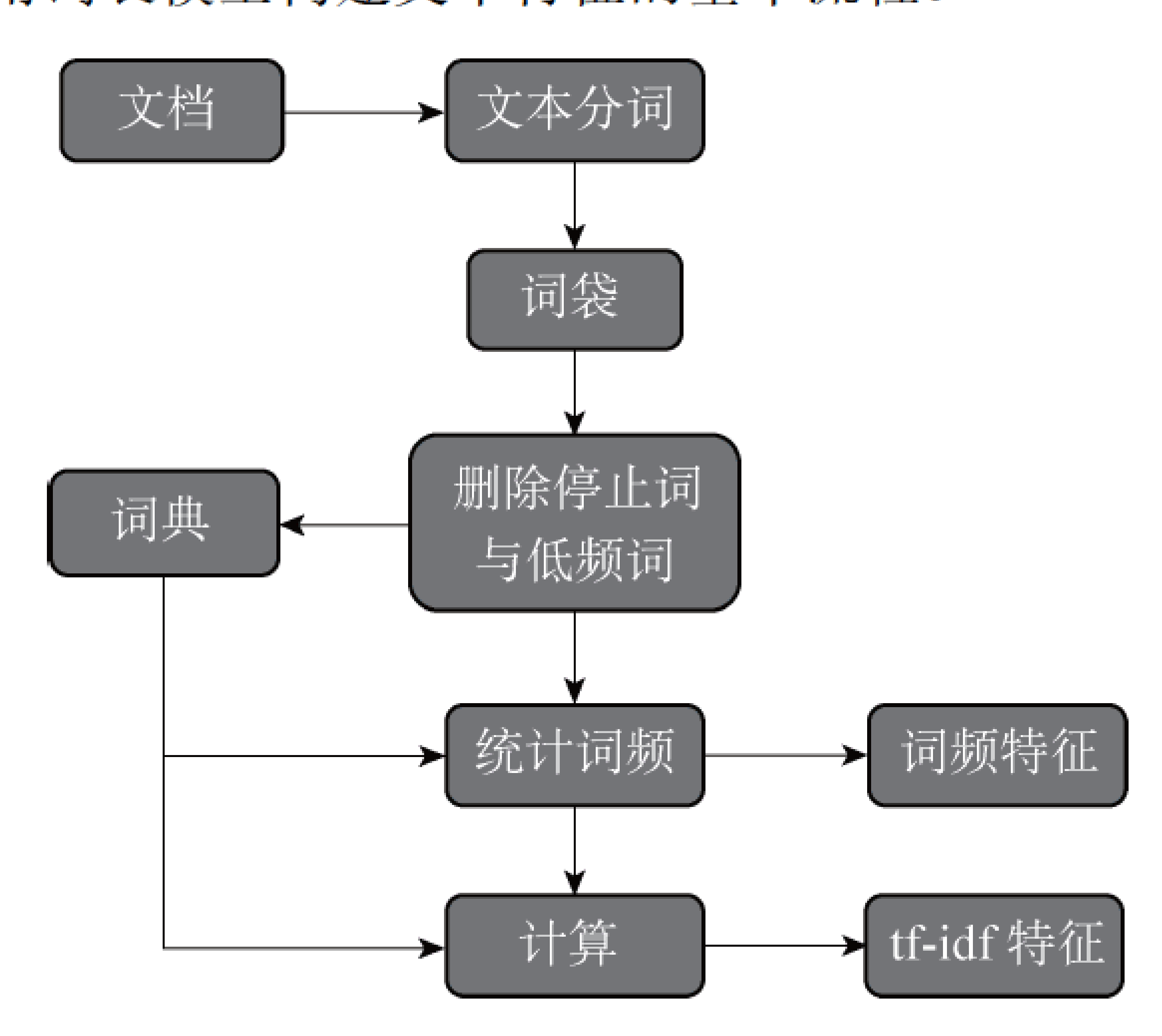


图 15.4　词袋模型应用的基本流程

提示：

停止词：词典中包含一些诸如“的”“也”“了”等词语。这些词语是构成中文句子的基本字词，无论文档围绕什么主题，这些词语都不可避免地大量出现，但却对区分不同文档的主题毫无帮助。类似于这样不携带任何主题信息的高频词称为停止词。

低频词：出现次数极低的词汇，通常是一些不常用的专有名词。它们可能出现在特定的文章中，但是并不能代表某一类主题。

15.7　案例：构建一个性别识别器

性别识别是一个有趣的问题。既然如此，我们将使用启发式的方法来构建一个特征向量，并且使用它训练一个分类器。这里使用的启发式是被给定名字的最后 N 个字母。例如，假设名字以 ia 结尾，它很可能是一个女性的名字，如 Amelia 或者 Genelia。另外，如果名字以 rk 结尾，它更可能是一个男性的名字，如 Mark 或者 Clark。由于不确定要使用的字母的确切数量，我们将使用这个参数来找出最佳答案。

假设我们要对 Sophia、Edward、William、Shirley 这四个名字进行分析，进而判断性别。我们用资料库中已有的训练数据来创建一个朴素贝叶斯分类器，再使用这个分类器对这些名字进行测试。我们把 N 的值分别设置为 1、2、3、4、5。使用 NLTK（自然语言处理工具包）中提供的内置方法来计算分类器的准确性。让我们来看看程序 15.6 是如何执行的。

程序 15.6　使用词袋模型提取词频矩阵：

```
import random
from nltk import NaiveBayesClassifier
from nltk.classify import accuracy as nltk_accuracy
from nltk.corpus import names

def extract_features(word, N=2):
    last_n_letters = word[-N:]
    return {'feature': last_n_letters.lower()}

if __name__=='__main__':
    male_list = [(name, 'male') for name in names.words('male.txt')]
    female_list = [(name, 'female') for name in names.words('female.txt')]
    data = (male_list + female_list)

    random.seed(5)
    random.shuffle(data)

    input_names = ['Sophia', 'Edward', 'William', 'Shirley']
    num_train = int(0.8 * len(data))

for i in range(1, 6):
```

```
    print('\nNumber of end letters:', i)
    features = [(extract_features(n, i), gender) for (n, gender) in data]
    train_data, test_data = features[:num_train], features[num_train:]
    classifier = NaiveBayesClassifier.train(train_data)

    for name in input_names:
        print(name, '-->', classifier.classify(extract_features(name, i)))

    accuracy = round(100 * nltk_accuracy(classifier, test_data), 2)
    print('Accuracy: ' + str(accuracy) + '%')
```

输出：

```
Number of end letters: 1
Sophia --> female
Edward --> male
William --> male
Shirley --> female
Accuracy: 74.7%

Number of end letters: 2
Sophia --> female
Edward --> male
William --> male
Shirley --> female
Accuracy: 78.79%

Number of end letters: 3
Sophia --> female
Edward --> male
William --> female
Shirley --> male
Accuracy: 77.22%

Number of end letters: 4
Sophia --> female
Edward --> male
William --> male
Shirley --> female
Accuracy: 69.98%
```

```
Number of end letters: 5
Sophia --> female
Edward --> female
William --> male
Shirley --> female
Accuracy: 64.63%
```

分析：

首先，我们导入了 random 模块，random 方法返回随机生成的一个实数，范围在[0,1)之间。接着我们从 NLTK 库中导入朴素贝叶斯分类器和测试分类器准确度的模块，最后导入语料库中的 names 模块。

在程序 15.6 中，我们首先定义一个函数，从输入的词汇中提取最后 N 个字母，程序的第 6～8 行实现了这一点。接着我们定义主函数，从 scikit-learn 包中提取训练数据，这个数据包含被标记的男性和女性的名字。程序的第 15～16 行是生成一个随机数字生成器并对数据进行洗牌。

提示：

seed()方法用于指定随机数生成时所用算法开始的整数值，如果使用相同的 seed 值，则每次生成的随即数都相同，如果不设置这个值，则系统根据时间来自己选择这个值，此时每次生成的随机数因时间差异而不同。shuffle()方法将序列的所有元素随机排序。

注意：

这两个方法是不能直接访问的，需要导入 random 模块，然后通过 random 静态对象调用该方法。

之后我们定义了一些用于测试的示例名称，并且定义了用于训练和测试的数据的百分比。数据的 80%将被用来进行训练。我们使用最后 N 个字符作为特征向量来预测性别，改变这个参数，我们看看准确度是如何变化的。程序的第 21～25 行是从 1～6 循环遍历参数 N 的不同长度来提取特征值，然后我们把这些特征值分为训练数据和测试数据，训练数据用来构建一个朴素贝叶斯分类器。程序的第 27、28 行是使用训练的模型预测输入数据的输出。最后我们使用 NLTK 中提供的内置方法来计算分类器的准确性并打印出来。

程序的输出显示了测试数据的准确性和预期结果。可以看到准确性在两个字母时达到最高，然后开始下降。

15.8　总　　结

在这一章中，我们学习了关于各种自然语言处理的基本概念。讨论了分词以及如何将输入文档分离成多个词。还学习了如何使用 stemming 和 lemmatization 将单词还原成基本形式。

我们还讨论了什么是词袋模型，并且为输入的文本构建了一个文档的单词矩阵，之后学习了怎样使用机器学习进行文本的分类。我们还使用启发式构建了一个性别识别器。

15.9　练　　习

（1）自己输入一些单词，分别使用 snowball stemmer 和 lemmatization 进行词形还原，观察输出并比较它们的区别（参考程序 15.2 和 15.3）。

（2）简述词袋模型的工作原理。

（3）自己定义一个包含两个主题的文本。

第 16 章　语音识别技术

在本章中，我们将认识语音识别技术。将要学习：

- ❑ 计算机感知声音。
- ❑ 计算机如何理解声音——频谱识别。
- ❑ 具体应用——语音识别。
- ❑ 基于 Python 的语音识别程序介绍。
- ❑ 语义理解。

16.1　计算机感知声音

在日常生活中想必大家都用过智能手机，很多智能手机中都有一个十分方便的功能，那就是通过语音下达命令来让手机执行，例如，我们对手机说“地图”，我们的手机就会打开它里面的地图 App，这就用到了语音识别的技术，运用这项技术的基础就是手机能够感知我们的声音。类似的，如果我们想对计算机下达命令让它打开某个文件夹，同样要使用到语音识别这项技术。那么现在问题就来了——我们都知道人类是通过耳朵感知声音的，那么手机或者计算机是如何感知声音的呢？

其实我们是通过对声波的一系列处理最终转化为便于计算机存储和处理的音频文件（MP3 格式等）。首先我们的话筒采集到声波通过传感器转换为电信号（如电压），就好比我们的耳朵里面的听觉感受器将声音传递给听觉神经。

但是不同于我们的大脑，计算机是无法识别连续信号的，所以我们只能将采样到的电信号变得离散（不连续），无论是在时间还是声波的幅度上都是如此，我们通过时间采样使得声波在时间上离散，同理我们通过分层的思想让声波的幅度也离散，这样我们连续的声波就转换成了离散的电信号，让我们的计算机可以识别，并且编码保存起来。这一系列的处理主要包括了采样、量化和编码等步骤（见图 16.1）。

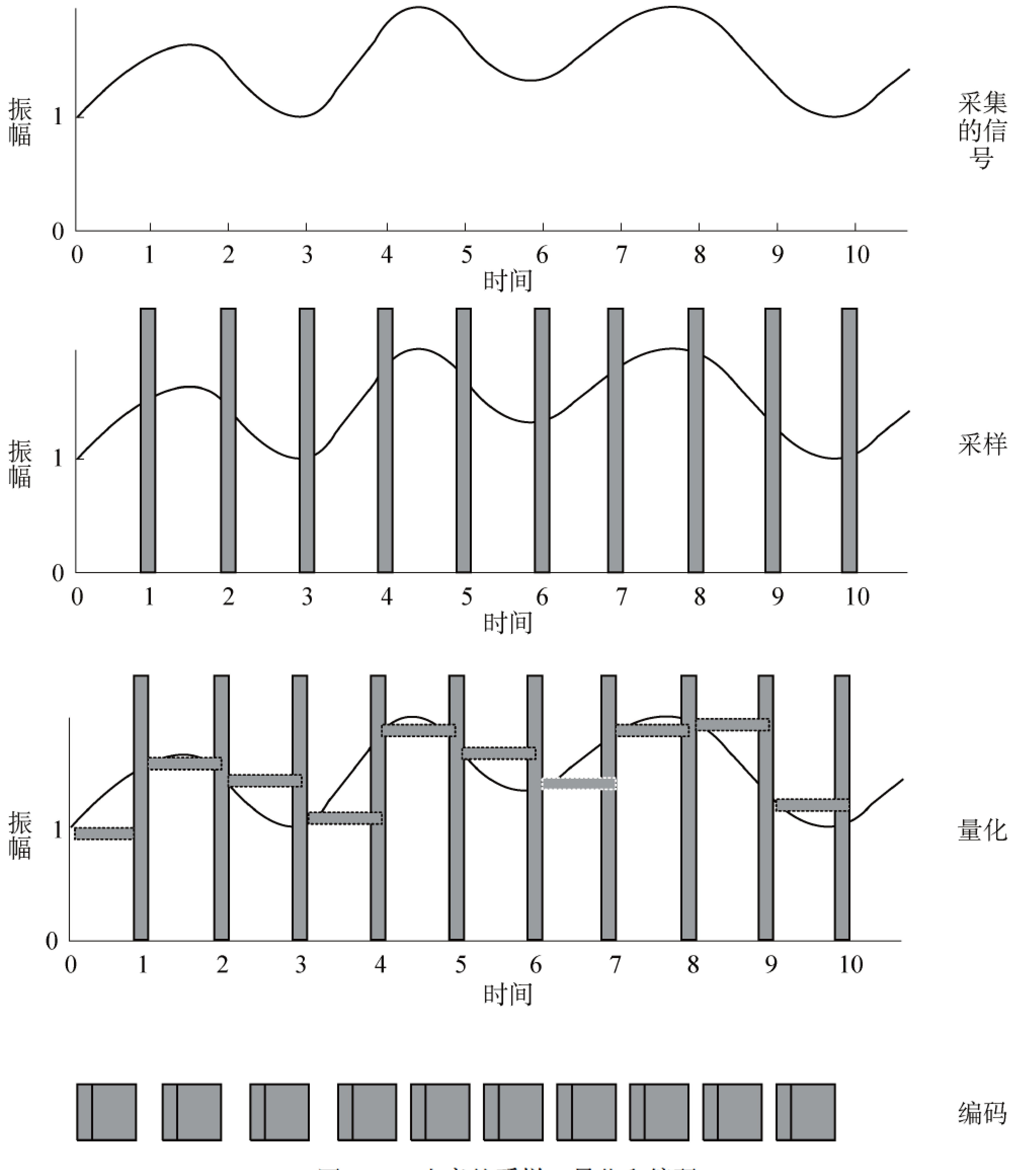


图 16.1　声音的采样、量化和编码

（1）采样：采样就是在某些特定的时刻对模拟信号进行测量，对模拟信号在时间上进行量化。具体方法是每隔相等或不相等的一小段时间采样一次。

（2）量化：分层就是对信号的强度加以划分，对模拟信号在幅度上进行量化。具体方

法是将整个强度分成许多小段。

（3）编码：编码就是将量化后的整数值用二进制数来表示。

通过上面的描述，我们大概了解了计算机是如何把自然界中的语音转换为音频文件的过程，下面我们通过录音程序 16.1 来进一步理解这个过程。

程序 16.1　录音小程序：

```
import wave
from pyaudio import PyAudio,paInt16
framerate=8000#采样率
NUM_SAMPLES=2000#缓存大小
def save_wave_file(filename,data):
    '''save the date to the wavfile'''
    wf=wave.open(filename,'wb')
    wf.setnchannels(channels)
    wf.setsampwidth(sampwidth)
    wf.setframerate(framerate)
    wf.writeframes(b"".join(data))
    wf.close()

def my_record():
    pa=PyAudio()
    stream=pa.open(format = paInt16,channels=1,
                   rate=framerate,input=True,
                   frames_per_buffer=NUM_SAMPLES)
    my_buf=[]
    count=0
    print("录音中：")
    while count<TIME*10:#控制录音时间
        string_audio_data = stream.read(NUM_SAMPLES)
        my_buf.append(string_audio_data)
        count+=1
        print('.')
    save_wave_file('01.wav',my_buf)
    stream.close()
chunk=2014

```

```
31:   if __name__ == '__main__':
32:       my_record()
33:       print('录音完成!')
```

输出：

```
控制台输出：
录音中：
.
.
.
.
录音完成！
文件路径输出：
该程序会在程序文件路径下生成一个音频文件——01.wav。
```

分析：

在上面的程序中，首先需要导入一个扩展库——PyAudio，打开“命令提示符”窗口输入 sudo apt-get install python-pyaudio python3-pyaudio 完成安装。PyAudio 是语音处理的 Python 库，它提供了录音、音频播放等许多音频处理的功能。

安装好之后就可以进行代码的编写了。首先确定音频的保存数据格式为 paInt16，录音的采样率为 framerate=8000，即 1s 内对声音信号的采样次数，录音缓存区大小为 NUM_SAMPLES=2000。然后我们定义录音的数据流 stream，通过 PyAudio 的实例对象 pa=PyAudio()来进行录音，程序的第 22～28 行是录音的控制流程，录音完成后调用音频保存方法 save_wave_file()将文件保存为.wav 文件。最后我们把数据流 stream 关闭 stream.close()，这样就完成了一次录音的过程。应该注意的是，我们的音频文件如果没有指定路径，则录音文件自动保存在和程序相同的文件夹下。

提示：

在学习 Python 的时候一定会使用到许多的扩展库，如果你的 Python 中缺少使用的库文件，不妨到 https://pypi.org/search/ 中去寻找想要的库文件。安装步骤参考上面的 PyAudio 库的安装步骤。

16.2　理解声音——频谱识别

通过对声波的处理，计算机可以感知声音并且可以进行编码储存了。就好比我们的大脑听到了某个声音并记了下来，接下来我们要做的就是识别出这是什么声音，是什么东西发出来的。那计算机到底如何“理解”声音的呢？

我们都听过很多歌手唱歌，有些歌手的声音低沉沙哑，有些歌手的声音嘹亮高亢。即使他们唱同一首歌，我们也能根据歌手的声音特色区分出是谁唱的。

那计算机是如何识别这些的呢？这里就需要计算机分析音频的依据——频谱，如图 16.2 所示。频谱的横坐标代表频率，纵坐标代表幅度——相应频率的声音对应的振幅。不同频率的声音占的能量多少，在频谱图上反映的就是频谱幅度的相对大小。例如，一段乐曲中的高音强低音弱，那么在一定范围内的频率高的区域频谱的振幅就大，反之，在频率低的区域对应的频谱幅度大。

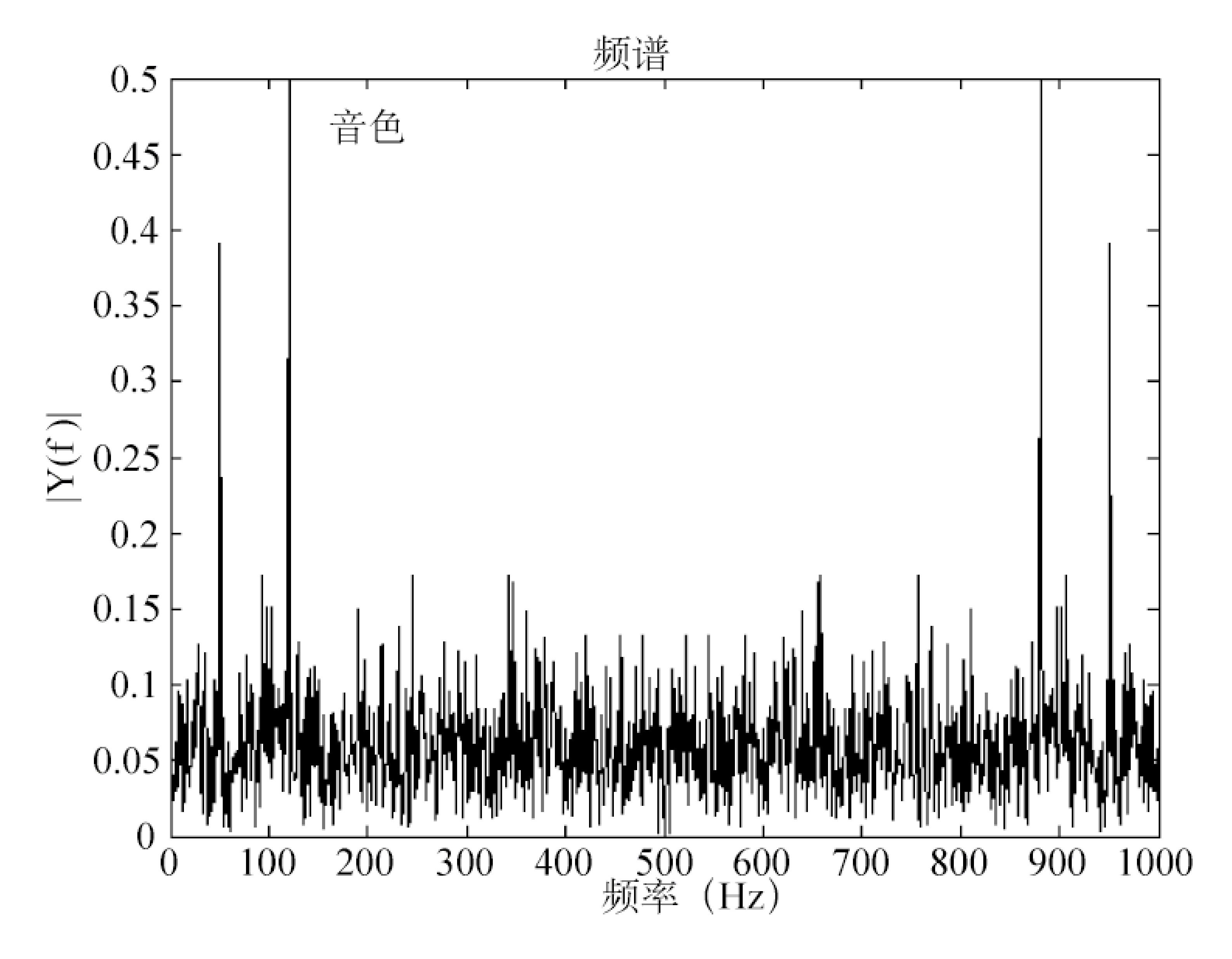


图 16.2　频谱图

物理课上学过声音的三要素——音调、响度、音色。基于这些要素，我们可以描述声音的特性。

（1）音调：表示了声音调子的高低，频率越高调子越高，反之频率越低调子就会越低，这部分可以用频谱来描述出来。

（2）响度：响度是我们常说的声音的大小，可以由波形的幅度来表示。

（3）音色：这是声音的一种更加复杂的特征，就算是相同的音调和响度，不同的乐器演奏或者不同的人来演唱都会有不同的效果。原因就是在不同的乐器和声带震动发声的过程中，除了发出音调对应的频率 F 之外，还伴着一些高频的成分（频率为 $2F$，$3F$，…），我们称之为泛音。这些高频的成分对应的幅度各不相同，带来了特别的听觉感受。这也就解释了为什么有些人即使唱同一首歌也会有不同的效果。

图 16.3 音色图中第一个最高峰所处的频率就是音调，而在这个频率的整数倍的位置都有不同大小的峰值，它们之间的比例反映了声音音色的不同。通过这些特性，我们就能大概分出这是什么发出的声音了。

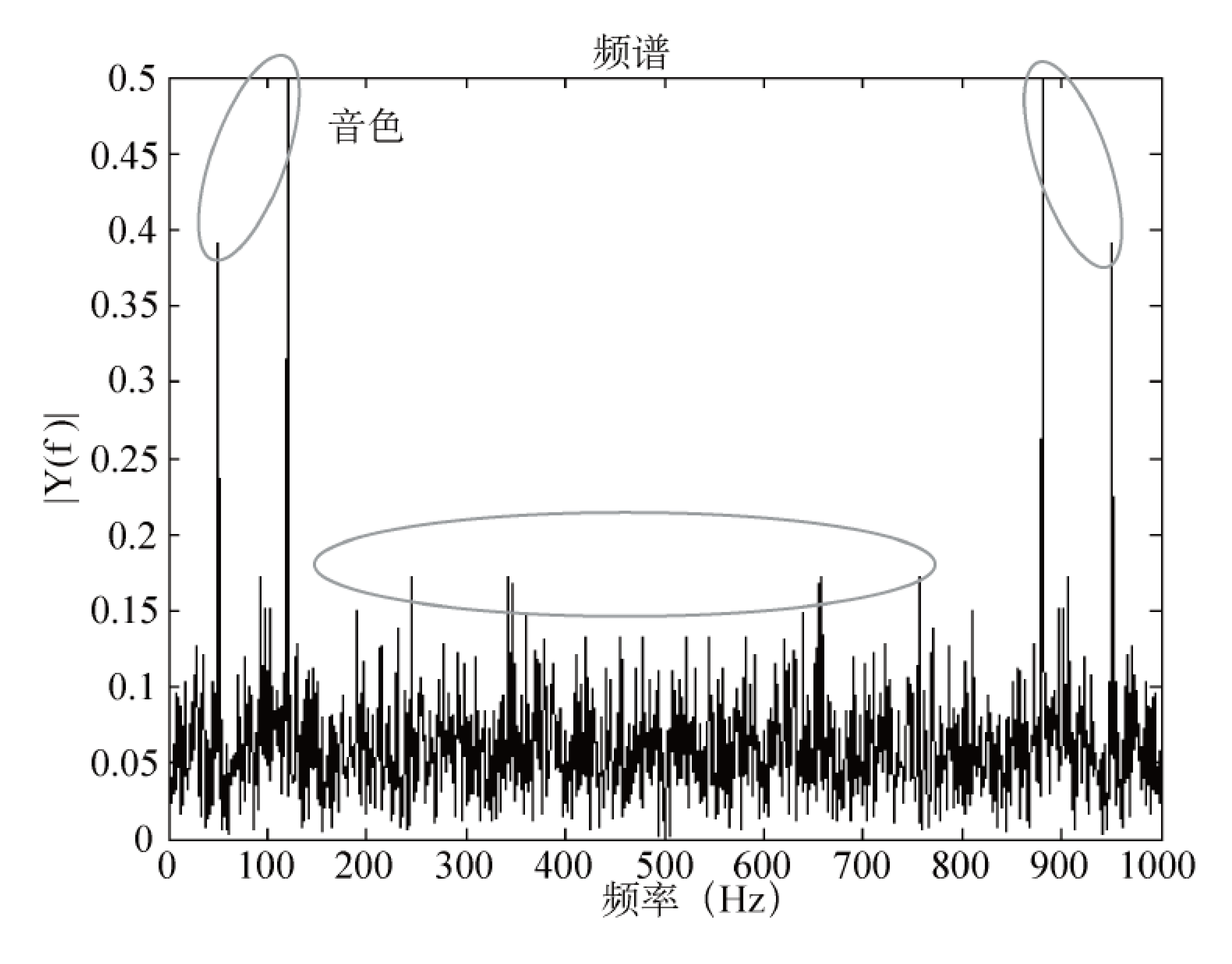


图 16.3　音色图

通过上面的文字和图像描述，我们知道了区分声音的重要依据是频谱，接下来通过程序 16.2 更加直观地体会下不同人读同一个字的频谱会有什么不同。

程序 16.2　频谱可视化：

```
import wave
import pyaudio
import numpy
import pylab
wf = wave.open("D:\\Python\\wavs\\Do-piano-2.20s.wav", "rb")
p = pyaudio.PyAudio()
stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
channels=wf.getnchannels(),
rate=wf.getframerate(),
output=True)
nframes = wf.getnframes()
framerate = wf.getframerate()
str_data = wf.readframes(nframes)
wf.close()
wave_data = numpy.fromstring(str_data, dtype=numpy.short)
wave_data.shape = -1,2
wave_data = wave_data.T
N=44100
start=0
df = framerate/(N-1)
freq = [df*n for n in range(0,N)]
wave_data2=wave_data[0][start:start+N]
c=numpy.fft.fft(wave_data2)*2/N
d=int(len(c)/2)
while freq[d]>4000:
    d-=10
pylab.plot(freq[:d-1],abs(c[:d-1]),'r')
pylab.show()
```

输出：

为了实现频谱的区别我们分别录下男女的“啊”声的音频——man.wav 和 girl.wav。

当打开的是 man.wav 文件时，输出的波形如图 16.4 所示。

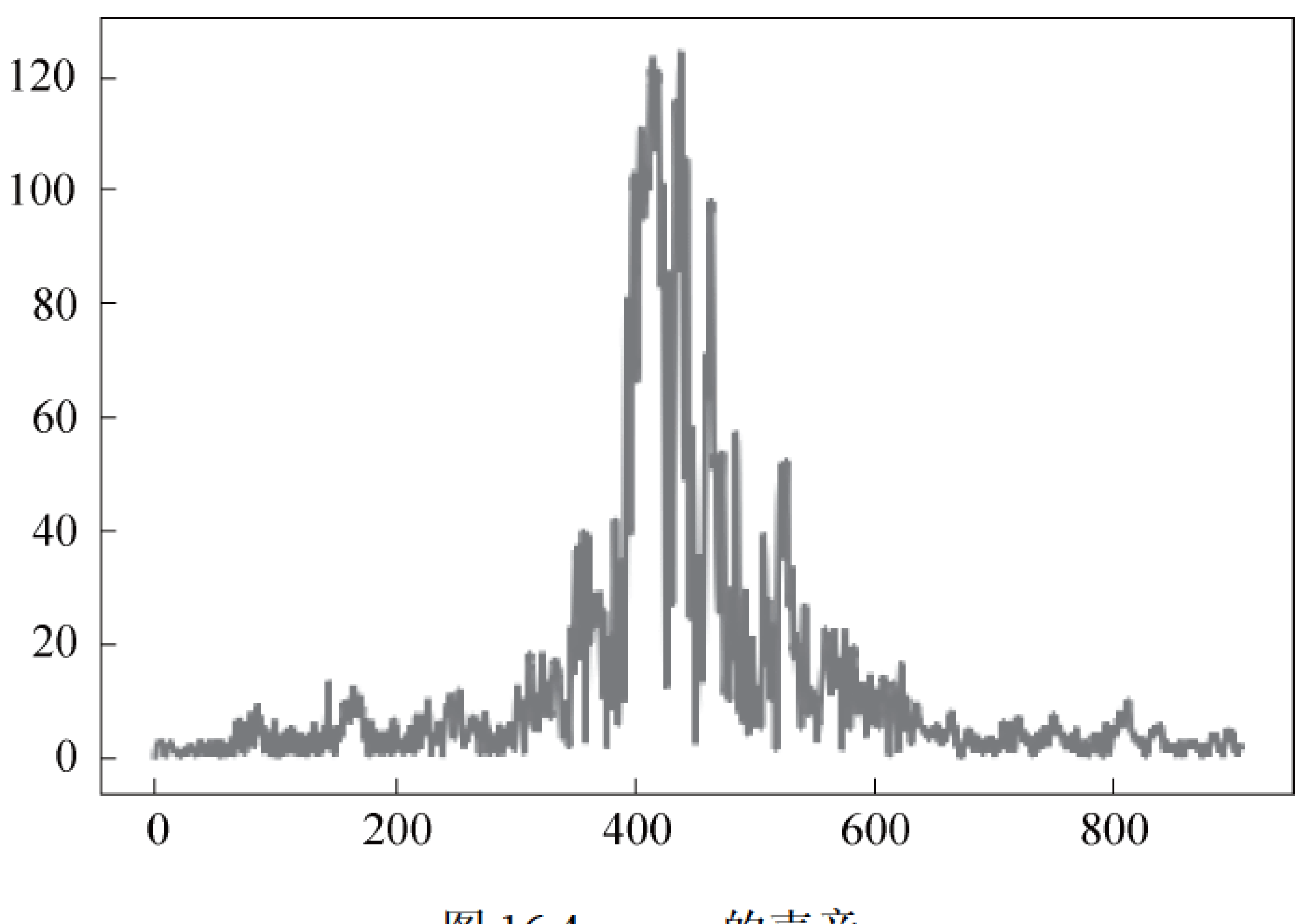


图 16.4　man 的声音

当打开的是 girl.wav 文件时，输出的波形如图 16.5 所示。

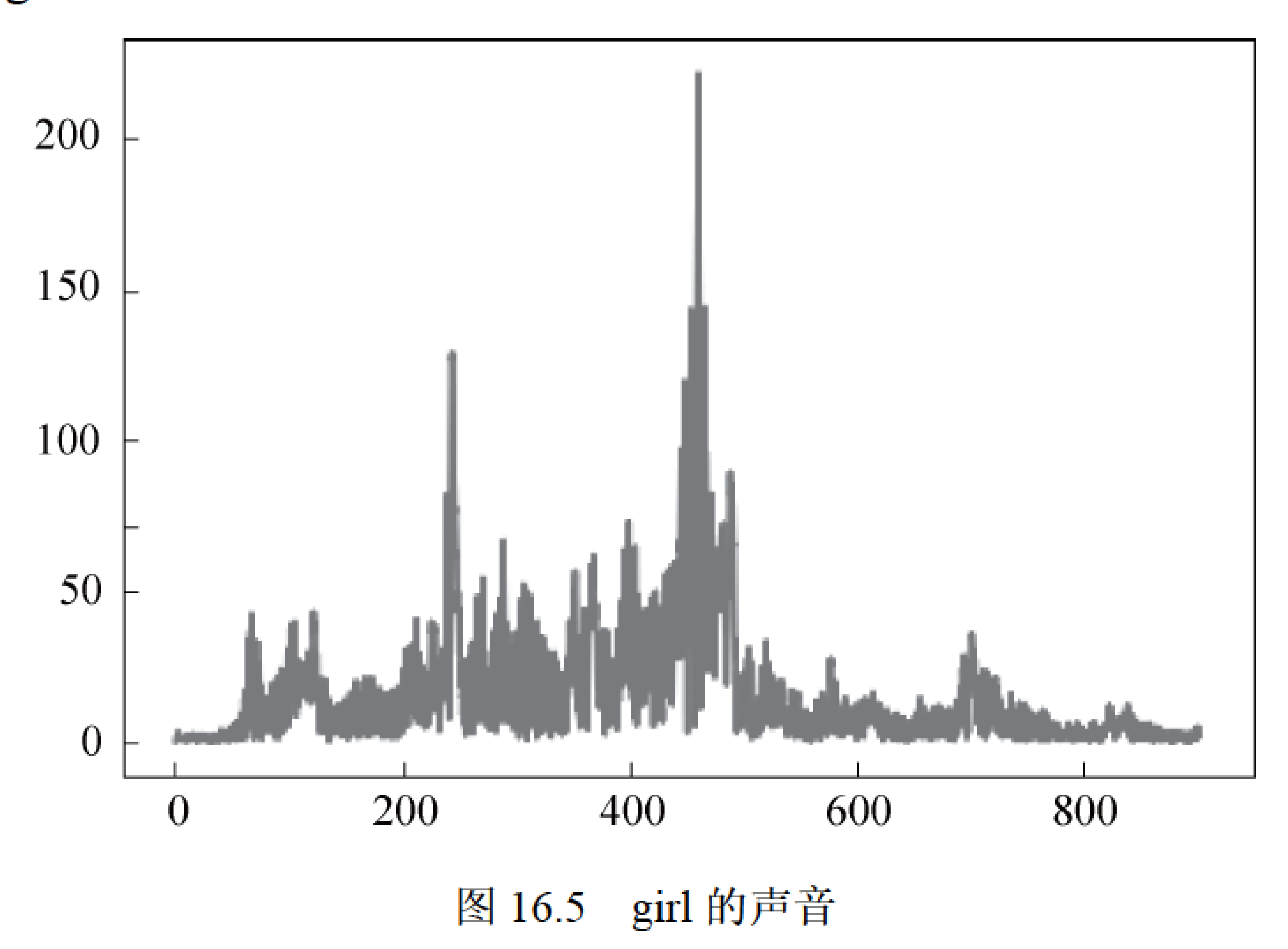


图 16.5　girl 的声音

分析：

在上面的程序我们还是调用了 PyAudio 库进行音频的处理，同时调用 pylab 库进行频谱图的绘制。

程序的第 5 行打开要可视化的音频文件，根据文件路径的不同，我们可以进行改动。程序的第 6 行我们实例化一个 pyaudio 对象。程序的第 7～14 行通过这个对象对音频进行操作。程序的第 13 行是将读取到的音频信息赋值给 str_data 方便后面的音频可视化操作。

程序的第 15～24 行是将音频信息转换成构建图形所需的信息，N=44100 表示采样的点数，越多表示越精确，start=0 是采样开始点，我们也可以修改采样的开始点，选取我们想要的某一段频谱。程序的第 25～28 行是频谱图的设置输出，通过 freq[d]>4000 可以设定输出的频率最大值，毕竟过高的频率是我们用不到的。这样的控制有利于去除无用的信息。通过打开不同的音频文件得到了不同的频谱图像，即使说的是同一个字，也能很容易地区分出这是两个不同的人说的。

16.3　语音识别原理

介绍了那么多计算机处理声音的知识，是时候考虑一下——我们学习这些知识到底用来干什么呢？举一个例子，我们都用过 QQ 或者微信，想必对语音转换并不陌生，在不方便听语音的时候往往会将语音转换成文字进行阅读。那计算机如何知道哪个声音对应哪个字呢？

为了解决这个问题，需要知道语音识别的原理。之前介绍了如何通过一整段音频独有的音调、响度、音色来区分是哪种声音。识别这样的特征少的分类任务其实并不难，但是语音识别却是对每一个音都进行一个分类，那么文字有多少我们的类别就有多少，这样看来即使是计算能力很强的计算机，执行这样的任务也是很耗费资源和时间的。

那么有没有好一点的方法在减少类别的同时又能保证准确度呢？其实人类的语言都是按照一定的规律组成的，如汉字都是通过拼音来确定读法的。这样看来区别拼音的声母（23 个）和韵母（24 个）比区别成千上万的汉字就简单很多了。不过仅仅这样我们的识别准确率还是不够，例如，nihao 计算机识别成“你好”还是“尼浩”呢？如果是我们来识别肯定是识别成“你好”，因为我们都学过了汉字的语言表达规律。如果让计算机也掌握这项能力，那么它识别汉字的准确度就基本能够满足我们的需求了。

我们通过图 16.6 语音识别过程图来展现语音识别的过程，语音识别技术是要识别每一个音的，所以需要将成段的语音分成若干小段，再将每个小段都识别成一个语音帧，再把这些帧根据声学特性组成声母和韵母，这就完成了我们语音识别的第一个阶段。也就是完成了识别 zhenhao 的过程，接下来我们就要确定它到底是“真好”还是“针好”了。要完成这个过程就需要通过语言表达规律即语言模型来判断最终的输出是符合我们预期的“真好”。

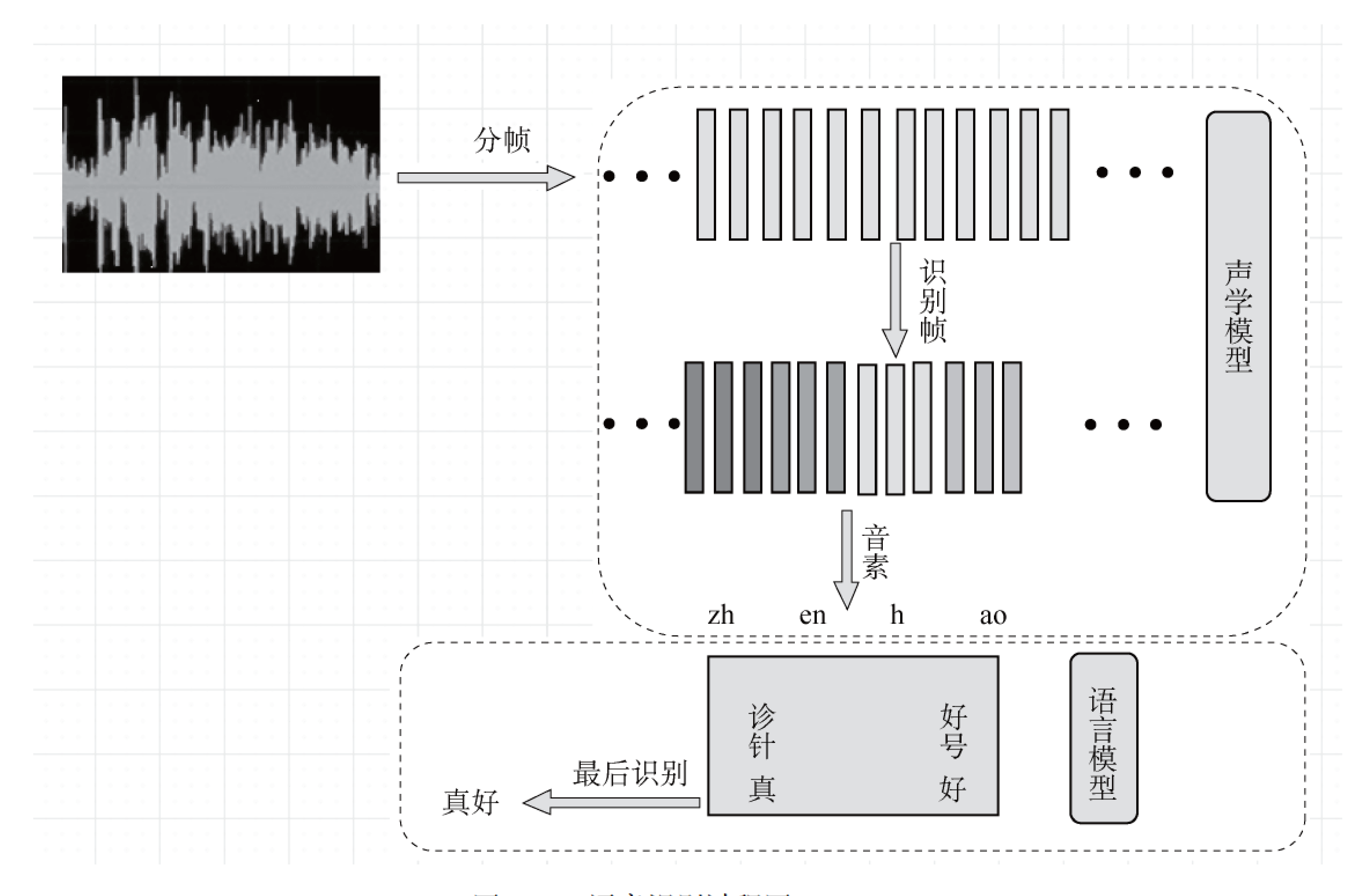


图 16.6　语音识别过程图

提示：

国内外语音识别技术较为先进的公司有国外的苹果、谷歌、微软、亚马逊、Facebook，国内的科大讯飞、百度语音、捷通华声、思必驰。这些公司通过自己的语音识别系统为我们的生活带来了许多的便利。

16.4　基于 Python 语音识别程序介绍

通过上面一节的介绍我们知道了语音识别的原理和过程，接下来将通过程序 16.3 来实现简单的语音识别功能。程序中我们通过调用百度的语音识别 API 完成最终的语音识别功能。百度语音 API 是一个免费语音识别接口，采用 HTTP 方式请求，可适用于任何平台的语音识别，而且准确率基本达到了我们正常生活所需的标准。在使用百度语音 API

之前我们要进行百度语音应用的注册用来获取 API_KEY 和 SECRET_KEY，只有获取这两个 KEY，百度语音 API 才能识别我们的身份，然后提供相对应的服务。识别的过程图如图 16.7 所示。

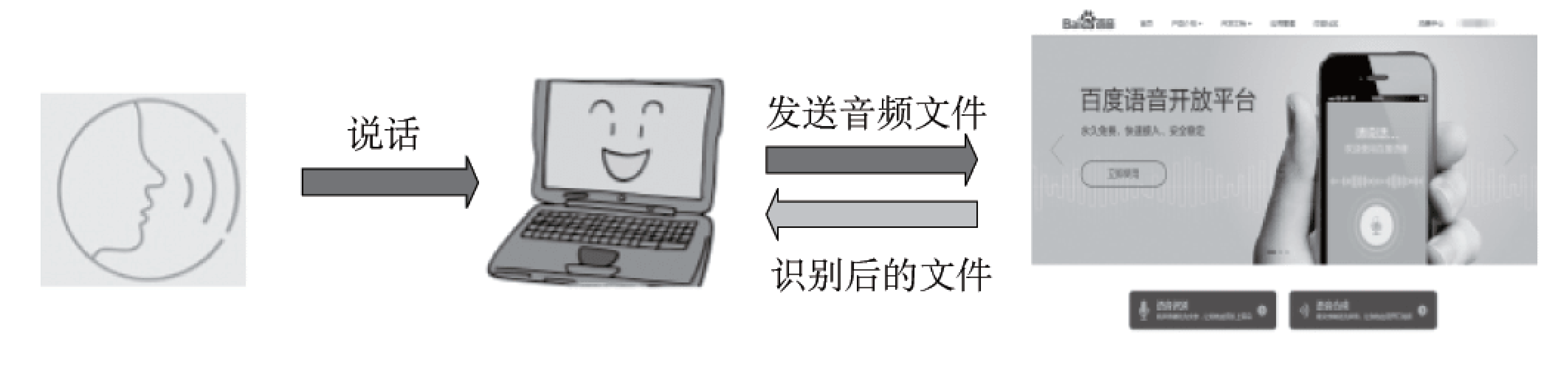


图 16.7　百度语音识别过程图

程序 16.3　语音识别：

```
import wave    #导入 wave 处理库
from pyaudio import PyAudio, paInt16
import time
from aip import AipSpeech
framerate = 16000 #采样频率，这里必须是 16000，不然百度语音识别不出来
NUM_SAMPLES = 2000 #一次录制的大小
channels = 1#录制的声道，1 为左声道，2 为右声道
sampwidth = 2
filepath = "F:/python3.5/语音文件 01/01.wav" #保存的路径，可以自定义
TIME = 2#时间单位

def save_wave_file(filename,data):
    wf=wave.open(filename,'wb')
    wf.setnchannels(channels)
    wf.setsampwidth(sampwidth)
    wf.setframerate(framerate)
    wf.writeframes(b"".join(data))
    eerro17: r-sequence-item-0-expected-str-instance-bytes-found
    wf.close()

def my_record():
    pa = PyAudio()
    stream = pa.open(format=paInt16, channels=1,
                     rate=framerate, input=True,
                     frames_per_buffer=NUM_SAMPLES)
```

```
    my_buf = []
    count = 0
    while count < TIME * 10:
        string_audio_data = stream.read(NUM_SAMPLES)
        my_buf.append(string_audio_data)
        count += 1
        print("正在录音：%d 秒" % count)
        save_wave_file(filepath, my_buf) #保存文件
        stream.close()
        chunk = 2014

def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

if __name__ == '__main__':
        #身份识别
    APP_ID = '11379632'
    API_KEY = 'QDeGx5sTzE1MsvojpvMltu41'
    SECRET_KEY = 'n9kRKRNX9hZ8eQDhdhcppbN77IQdCHgo'

    client = AipSpeech(APP_ID, API_KEY, SECRET_KEY)
    print("准备开始录音.......3")
    time.sleep(1)
    print("准备开始录音.......2")
    time.sleep(1)
    print("准备开始录音.......1")
    time.sleep(1)
    print("开始录音.......")
    my_record()
    print("正在识别，请稍后.......")
    str = client.asr(get_file_content('F:/python3.5/语音文件/01.wav'), 'wav', 16000,
          {'dev_pid': 1536, })
    print("结果为：" + (str["result"][0]))
    print("*********************************************")
```

输出：

```
准备开始录音.......3
准备开始录音.......2
准备开始录音.......1
```

```
开始录音.......
正在录音：1 秒
正在录音：2 秒
正在录音：3 秒
正在录音：4 秒
正在录音：5 秒
正在录音：6 秒
正在录音：7 秒
正在录音：8 秒
正在录音：9 秒
正在录音：10 秒
正在识别，请稍后.......
结果为：你好我是小明
**********************************************
```

分析：

在这个程序中我们主要用到两个 Python 库，第一个是 PyAudio，它是用来进行音频处理的，只有导入这个库才能实现我们的录音功能。第二个是程序第 4 行导入的百度 aip，安装的方法是在命令提示符中输入 sudo pip install baidu-aip。

导入这些包是我们程序建立的基础，库的导入和安装参考 16.1 节里面 PyAudio 库的安装过程。通过录音并将录音文件保存，再将保存的音频文件上传到百度语音 API 并将识别后的结果返回，最后我们将得到的结果进行处理，输出想要的结果。程序的第 12～19 行定义了文件的保存方法，因为需要录音，那么保存成什么文件还有保存的路径在哪里我们都要定义好。程序的第 21 行定义了录音方法，第 28 行实现的是录音时间的控制，第 29 行实现的是录音采样的缓存大小设置，然后调用文件保存方法把我们的声音保存成音频文件。程序的第 37～39 行实现的是将保存音频文件读出来。通过第 57、58 行的代码发送到百度 AIP，最后在第 59 行输出识别的结果。

提示：

因为是调用百度接口来进行语音识别，所以我们需要进行百度语音识别的应用申请，完成申请后会获得类似程序的第 44 和 45 行的 API Key 与 Secret Key，只有这样我们才能调用百度接口进行识别。具体的步骤可以浏览网址 https://jingyan.baidu.com/article/f3e34a12df0cddf5eb65359f.html。

16.5　简单语义理解

程序 16.3 中我们说“你好，我是小明”，通过百度语音识别了之后，相当于我们知道对方说的话，然后怎么理解这句话的意思就是我们语义理解的关键了。在我们日常与别人对话的时候，除了听到别人的话之外，最重要的还是理解别人的意思。只有这样我们才能与别人进行交互，例如，别人说“你喜欢什么？”，我们就会回答“我喜欢……”。我们通过语句中的关键词“你”“喜欢”来理解对方的意思——他问我喜欢的东西，而不会理解为他问我小明喜欢的东西。让计算机根据关键词的判断来进行语义的理解是行得通的，这个过程如图 16.8 所示。

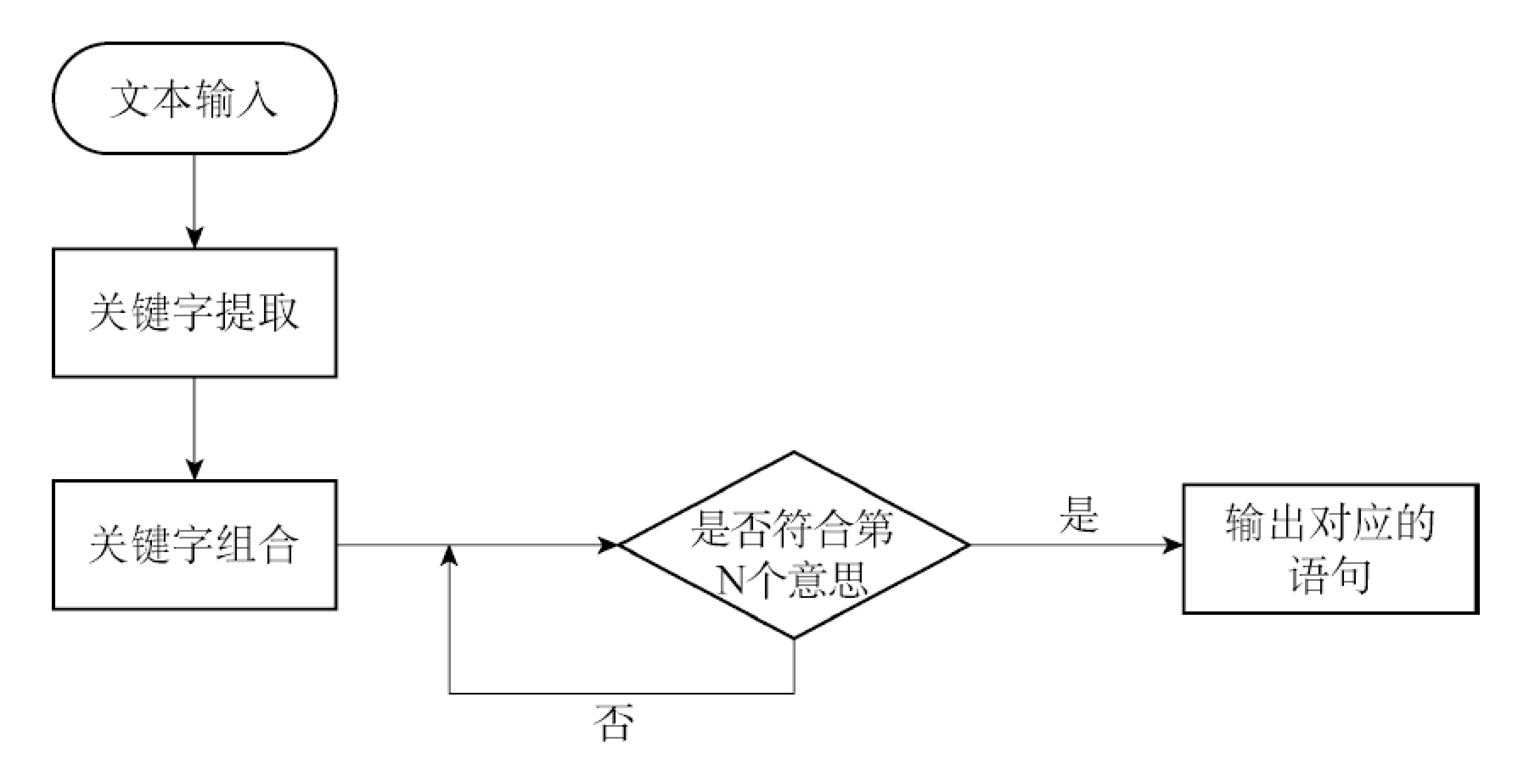


图 16.8　简单语义识别的基本流程

下面通过程序 16.4 简单了解下什么是语义的理解。

程序 16.4　简单语义理解：

```
print("请输入你想说的话：")
text = input()
#通过 find()查询文本中是否含有某些词，如果找到返回 0
you=text.find('你')
good=text.find('好')
say= text.find("叫")
weather=text.find("天气")
```

```
7:   handsome=text.find("帅")
8:   beautiful=text.find("漂亮")
9:   who=text .find("谁")
10:  know=text.find("知道")
11:  like=text.find("喜欢")
12:
13:  if (you is 0 & good is 0 & say is 0):
14:     print('你好啊，我是人工智能')
15:  elif (you is 0 & handsome is 0):
16:       print("一般一般，世界第三啦")
17:  elif (you is 0 & beautiful is 0):
18:           print("你也挺漂亮的哈哈"):
19:  elif weather is 0:
20:         print("还好啦，反正我还感觉不到")
21:  elif (you is 0 & know is 0):
22:         print("很抱歉啊，我还小，很多事都不知道呢")
23:  elif (you is 0 & like is 0):
24:       print("还好啦，我也并没有特别喜欢什么")
25:  else:
26:       print("你说啥？再说一遍呗")
```

输出：

```
请输入你想说的话：
天气挺好的
还好啦，反正我还感觉不到
```

分析：

在程序 16.4 中，可以通过第 1、2 行进行一个文本的输入。然后在第 3～11 行通过 find()方法寻找到我们需要的词语,并赋值给变量，如果在输入的文本中找到我们想要的词语会返回 0。

这时候可以再通过判断该变量是否为 0，就可以知道文本中到底有没有我们想要的词语。例如，程序的第 3 和第 4 行查找是否存在“你”和“好”，如果含有会返回 0。我们通过第 13 行判断 you 和 good 是否为 0 决定是否输出第 14 行的内容。当然输入的文本符合第 13～25 行中的任意一个判断语句就会输出相对应的内容。这种通过变量组合来判断输出是计算机实现语义理解的一个有效方式。

16.6　总　　结

在这一章中，我们学习了计算机是将声音信号离散并将声音进行存储识别的，也知道了从声音的三要素——音调、音色、响度，来区别声音的差异。除此之外介绍了语音识别的原理，我们建立语音模型降低语音识别的识别难度，又增加了语言模型提高了识别的准确率，两个模型一起使用，让语音识别功能基本达到了我们想要的准确度。我们调用百度语音识别接口实现了该功能，在语音识别的基础上，我们将进行简单的语义理解，让程序执行相应的命令。

16.7　练　　习

（1）通过程序 16.1 录制两段音频，再通过程序 16.2 生成频谱图并区别两者的差异。

（2）在程序 16.4 中定义自己的变量组合，输出自己想要的话。

（3）利用程序 16.3 说几段话，看识别的准确率。

第 17 章　计算机视觉

在本章中，我们将了解人工智能的一个重要分支——计算机视觉，并学习利用计算机视觉解决一些问题。将要学习：

- 认识并了解计算机视觉的相关知识。
- 简单了解 PIL 类库并学习基础的图像处理知识。
- 学习 OpenCV 的基础知识以及简单应用。
- 利用背景差分法捕捉视频中的移动物体。
- 简单学习利用颜色空间进行目标跟踪。
- 简单了解人脸识别系统的工作原理和步骤。

17.1　计算机视觉简介

看过电影《钢铁侠》的读者一定会记得主角斯托克的人工智能管家——贾维斯，他具有很高的智能，可以帮助斯托克处理各种事，如设计图纸、组装完成“钢铁侠”等。在《钢铁侠 1》中有一个小细节，钢铁侠初次飞行，他的人工智能系统在头盔中显示出了道路上的车辆信息，包括车辆的品牌、车中的人物等。虽然是电影中的情节，但很有趣的是，这确实就是人工智能中计算机视觉的应用，而且这也不再是科幻，现今计算机视觉技术发展的程度已经完全可以做到。

有人说，计算机视觉是人工智能的一扇大门，因为对于人来说，视觉的反馈往往更加重要，人的大脑皮层中有 70%都在处理视觉信息，没有视觉，人工智能将只会是一个空架子。现如今，计算机视觉早已成为研究人工智能的一个重要领域，也在各种领域发挥着独有的作用。例如，军事上的导弹巡航系统、交通上的道路监控系统、医学影像处理、应用在各大企业单位的人脸识别系统以及现在非常热门的 VR（Virtual Reality，虚拟现实）全息，全都离不开计算机视觉。

随着时代的发展，计算机技术日新月异，2008 年第一部《钢铁侠》上映的时候我们惊

叹于机器人管家的高智能，然而随着技术的不断深入，到了今天，人工智能管家已经不再是故事中的事物，即便是造不出一个真正的钢铁侠，但是制造一个可以处理日常琐事的人工智能管家却不再是个难以企及的梦想。也许到了未来，人工智能管家会像智能手机一样普遍，计算机视觉也将更加广泛地运用到各行各业，这需要我们共同的努力，也许现实版的“钢铁侠”就在你的手中诞生。

在这一章，我们将学习计算机视觉方面的相关内容，通过一些简单的例子来加深我们对计算机视觉以及人工智能的理解。下面，就让我们开始学习计算机视觉的相关知识。

17.2　图像的操作与处理

大家都知道，平时我们所看到的流畅视频画面是由一帧一帧的图像构成。当我们需要对一个视频文件进行分析时，连续播放的视频并不那么容易进行采样分析，所以在计算机视觉中对视频的分析通常都是对视频帧的分析。计算机视觉就是一门研究如何对图像中的信息进行自动提取的学科。

学习计算机视觉首先让我们学习如何对图像进行处理，这里我们用到了 PIL 图像处理类库，它提供了我们通常使用的各种图像处理功能，如缩放、裁剪、旋转、颜色转换、模糊等。PIL 官方版本均是以 Python 2.X 的版本发行的，并没有针对 Python 3.X 的 PIL 版本，这里我们可以使用 Pillow 库，它是一群志愿者在 PIL 的基础上建立的兼容版本，而且增添了很多新的功能。推荐大家下载安装 Anaconda，它是一个开源的 Python 包，包含了 180 多个库，我们研究图像操作所使用的 Pillow、numpy、matplotlib 等库均包含在内，直接在“命令提示符”窗口中利用 pip install 命令进行安装即可。

接下来，通过程序 17.1 来简单了解图像处理的基本操作。

程序 17.1　图像处理基本操作：

```
from PIL import Image
import matplotlib.pylab as plt

img1=Image.open('img.jpg')
img2=img1.convert('L')
img2.resize((128,128))

```

```
plt.subplot(2,1,1)
plt.imshow(img1)
plt.subplot(2,1,2)
plt.imshow(img2)
plt.show()
```

输出，如图 17.1 所示：

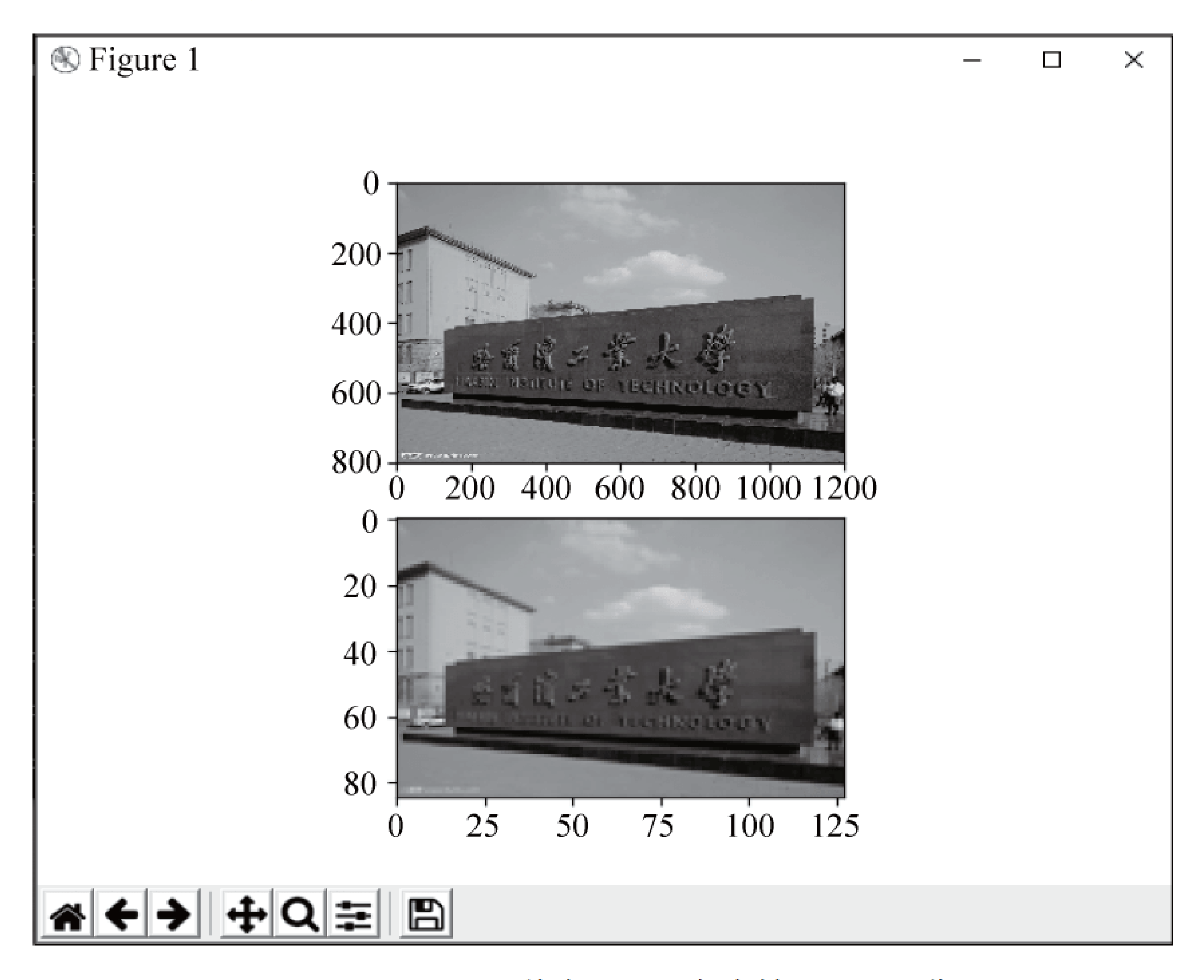


图 17.1　原图与重新调整大小以及灰度转化后的图像对比

分析：

PIL 库中比较重要的一个模块就是 Image，它包含了图像的读取、保存等函数。例如，程序 17.1 中使用的 open()函数，这很容易理解，不管我们要做什么处理，都是要先打开图像才能进行后续的处理。我们还用了 convert()函数将图像转换成灰度图，因为在很多时候，图像本身的颜色会对我们所要分析的信息造成干扰，而灰度图可以帮助我们更好地分析问题。最后还使用了 resize()函数调整了图像的大小，将其调整为 128 像素×128 像素大小的图，在输出中，我们可以看到第二张图的坐标轴已经改变。

在程序 17.1 中我们还引入了 matplotlib 库，它具有十分强大的绘图功能。subplot(2,1,1)语句代表我们要将图片输出成两行一列的第一行，这样可以使一个窗口中显示多幅图片，imshow()函数在内存中绘制图像，并不显示，要用 show()函数打开图像用户界面，向用户显示图像。

很多时候我们在对一个图像进行分析时，图像的像素值通常是一个很好的着手点，而图像的直方图则可以比较直观地展示图像的像素分布。通过绘制图像直方图，我们可以进行图像检索、图像分割、图像分类等。

图像的直方图与数学中的直方图类似，将像素值范围划分成一定数目的小区间，每个小区间内就是落在该区间表示范围内的像素数目，用这种形式来表征图像中颜色的分布情况。程序 17.2 将绘制一幅图像（见图 17.2）的直方图。

程序 17.2　绘制图像直方图：

```
from PIL import Image
from pylab import *
import numpy as np

img=np.array(Image.open('img.jpg').convert('L'))
figure()
hist(img.flatten(),128)
show()
```



图 17.2　原图

输出，如图 17.3 所示：

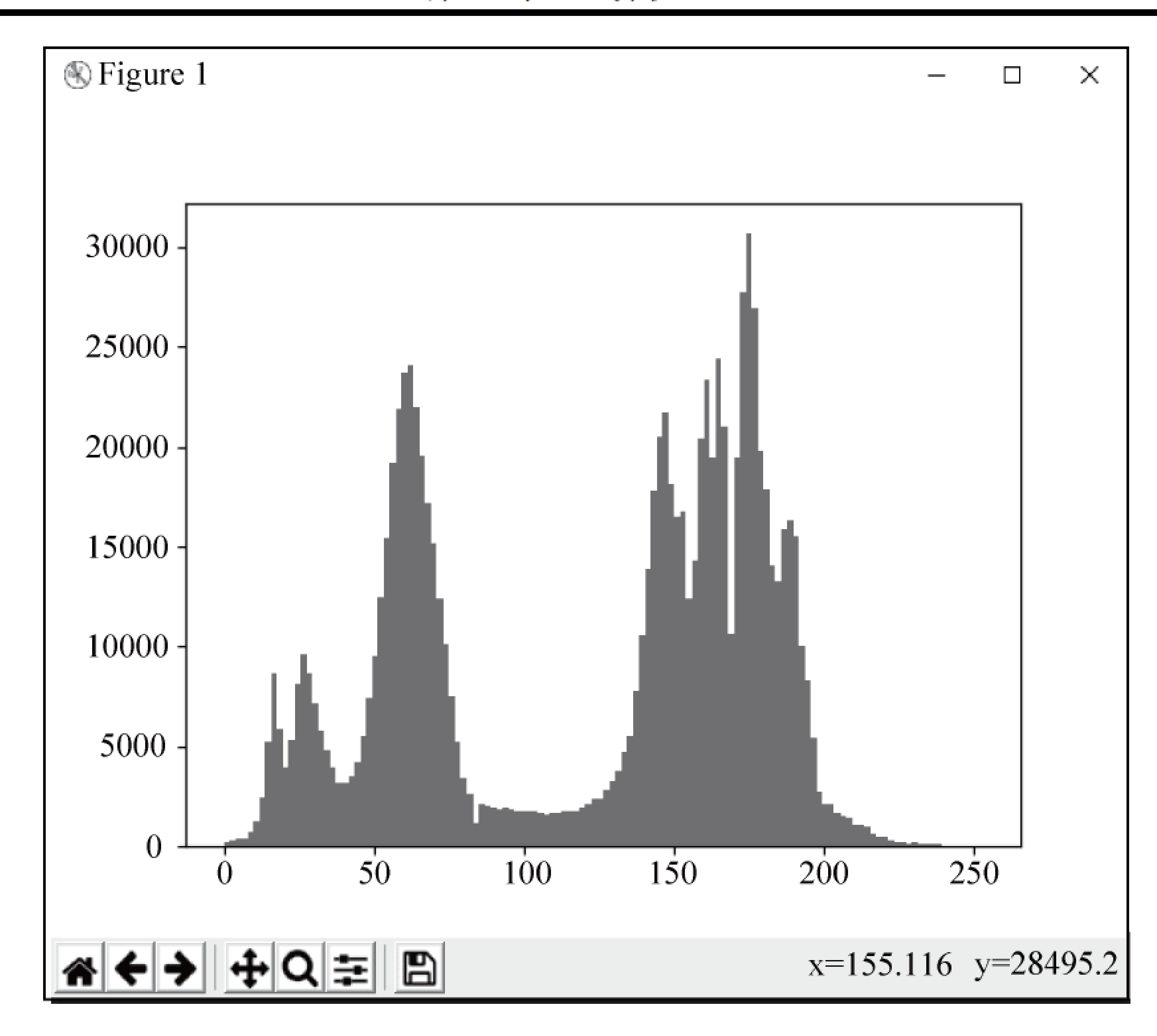


图 17.3　原图对应的直方图

分析：

如程序 17.2 所示，我们导入了一个新的库——numpy，它是一个使用非常广泛的科学计算工具包。我们使用 numpy 库的 array()函数将图像转化成一个数组对象，用三元组表示像素点——（行、列、颜色通道）。直方图的绘制使用的是 hist()函数，函数的第一个参数是输入的图像，第二个参数指定划分的小区间的数目。

注意：

hist()函数只接受一维数组作为输入，所以我们需要对输入图像做一些转换。这里我们用到了 flatten()函数，flatten()函数将数组按照行优先原则转化成一维数组，再利用 hist()函数绘制即可。

计算机视觉中对于图像的操作还有很多，如旋转、去噪、模糊、扭曲等，这里只介绍最简单的两种操作，感兴趣的同学可以自行查找资料，了解关于计算机图像处理的各种操作算法。下面我们将学习一个新的内容——OpenCV，了解这个计算机视觉类库包含了哪些操作。

17.3　OpenCV 的基础知识

计算机视觉的学习离不开 OpenCV，它实现了很多图像处理以及计算机视觉方面的算法，如物体跟踪、探测，人脸识别，视觉探测与追踪、特征提取等计算机视觉方面的应用。OpenCV 作为一个开源的计算机视觉类库，可以在多种操作系统上运行，同时也提供了大量的 Java、Python、MATLAB 接口，致力于真实世界的实时应用。

OpenCV 是由一系列 C 函数和少量的 C++类构成的，包含了计算机视觉领域的很多模块。OpenCV 最初由英特尔公司开发，在 2008 年获得了在当时十分具有影响力的机器人公司——Willow Garage 的支持，不幸的是，作为业界传奇的 Willow Garage 机器人公司于 2014 年破产。OpenCV 在 2012 年被非营利组织 OpenCV.org 接管维护。

Python 作为一种简洁明了的脚本语言，在 C++基础上的 Python 接口得到了越来越广泛的支持。我们可以根据自己所安装的 Python 版本选择所要安装的 OpenCV 版本，不同的操作系统对应的 OpenCV 不同。在安装时，要对下载的 whl 文件进行重命名，否则就会报错。选择下载的文件如图 17.4 所示。

opencv_python-3.4.1-cp35-cp35m-win32.whl
opencv_python-3.4.1-cp35-cp35m-win_amd64.whl

图 17.4　OpenCV 版本选择

opencv python-3.4.1 是 OpenCV 的版本，cp35 指的是计算机中所安装的 Python 版本是 3.5，Python 版本可以在“命令提示符”窗口中输入 python –version 进行查询（version 前是两个-），根据计算机系统的位数选择 win32（32 位）或者 amd64（64 位）。安装前要将文件重命名成 opencv_python-3.4.1-cp35-none-win_amd64.whl。重命名后在“命令提示符”窗口输入 pip install opencv_python-3.4.1-cp35-none-win_amd64.whl 进行安装，安装后进入 Python 命令行中输入 import cv2 检验是否安装成功。出现如图 17.5 所示界面即为安装成功。

```
C:\Users\69033>python
Python 3.5.4 (v3.5.4:3f56838, Aug  8 2017, 02:17:05) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>>
```

图 17.5　在“命令提示符”窗口中检验 OpenCV 是否安装成功

OpenCV 作为一个计算机视觉类库，很多时候需要对动态视频进行分析、计算和处理。

OpenCV 能够很好地支持从摄像头读取视频，通过对摄像头读取的视频帧进行分析，达到最终需要的物体探测或者跟踪的结果。如程序 17.3 所示是一个简单的小案例，用来打开摄像头并将图像变成灰度图。

程序 17.3　打开摄像头捕获视频帧：

```
import cv2

cap=cv2.VideoCapture(0)
while True:
    ret,frame=cap.read()
    frame=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    cv2.imshow('Output',blur)
    key=waitKey(10)
    if key == 27:
        break
```

输出，如图 17.6 所示：

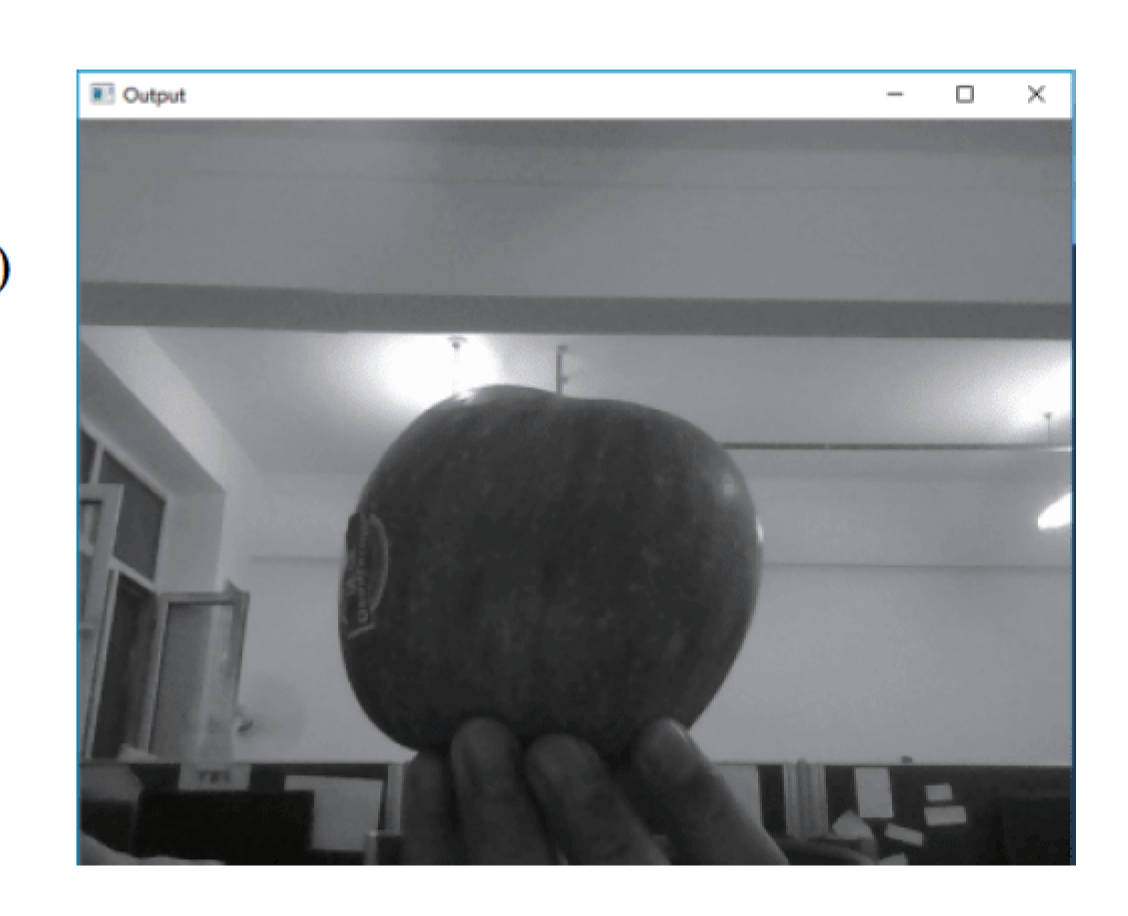


图 17.6　对获取的视频帧进行灰度处理

分析：

导入 OpenCV 的 cv2 模块，利用 VideoCapture()函数可以打开一个本机文件夹内的视频文件，也可以利用整数来启动摄像装置，0 代表计算机默认摄像头。打开视频/摄像头后，看到的是一个连续的动态的画面，这很不利于我们分析问题，所以对视频的处理需要逐帧进行，这里我们使用 while 循环来实现对视频帧的逐一提取。在计算机视觉的研究过程中，很多时候我们最常用的 RGB 颜色模型并不好用，所以要利用 cv2 模块的 cvtColor()函数进行颜色转换，参数 cv2.COLOR_BGR2GRAY 是将图像转化为灰度图，除此之外 HSV 颜色模型的转化也很常用。最后我们使用了 waitKey()函数，这个函数可以监控键盘操作，27 是 ESC 键的 ASCⅡ码值，故只要按下 ESC 键就可以关闭摄像头退出当前程序。下面，我们用流程图 17.7 来加深对这个程序的理解。

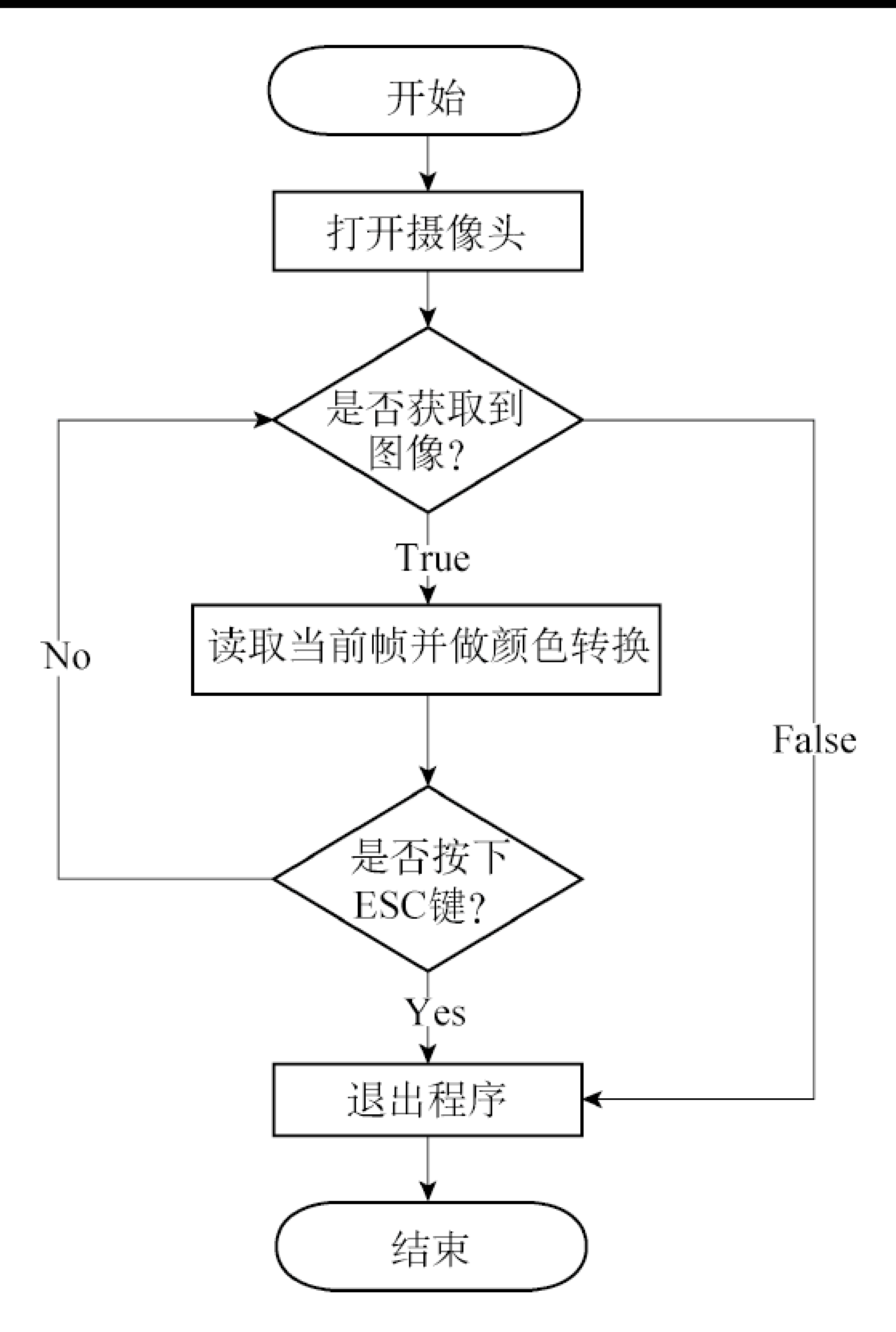


图 17.7　对视频帧进行基本操作的流程图

注意：

OpenCV 中，图像的颜色模型不是传统的 RGB，而是按照 BGR 顺序存储的。颜色空间的转换函数 cvtColor()的几个常用参数有 cv2.COLOR_BGR2GRAY（转化成灰度图）、cv2.COLOR_BGR2HSV（转化成 HSV 颜色空间）。

17.4　背景差分法检测物体

在计算机视觉的实际应用中，我们发现通常要想实现一个功能，最基本的就是先要找到需要分析、计算的目标，例如，在道路监控系统中，摄像头拍下的视频画面中包含道路、道路两侧的公共设施、车辆，其中车辆又包括小轿车、货车、自行车等，而我们最主要的就是需要监

控机动车的行驶状况，所以首先我们要让计算机在视频中找到机动车，而忽略无关物体。再例如，现在很多企业、单位采用人脸识别系统作为出入门禁、上下班打卡的方式，人脸识别系统也是通过摄像头捕捉到人脸，再将捕捉到的人脸图像与数据库比对来进行身份认证。

背景差分法是在一个静止的场景下对移动物体进行追踪的方法，以背景做模型，然后在视频帧中减去这一背景模型，获得的前景图像就是我们需要追踪的物体。我们用流程图 17.8 来看一下背景差分法跟踪目标物体的过程。

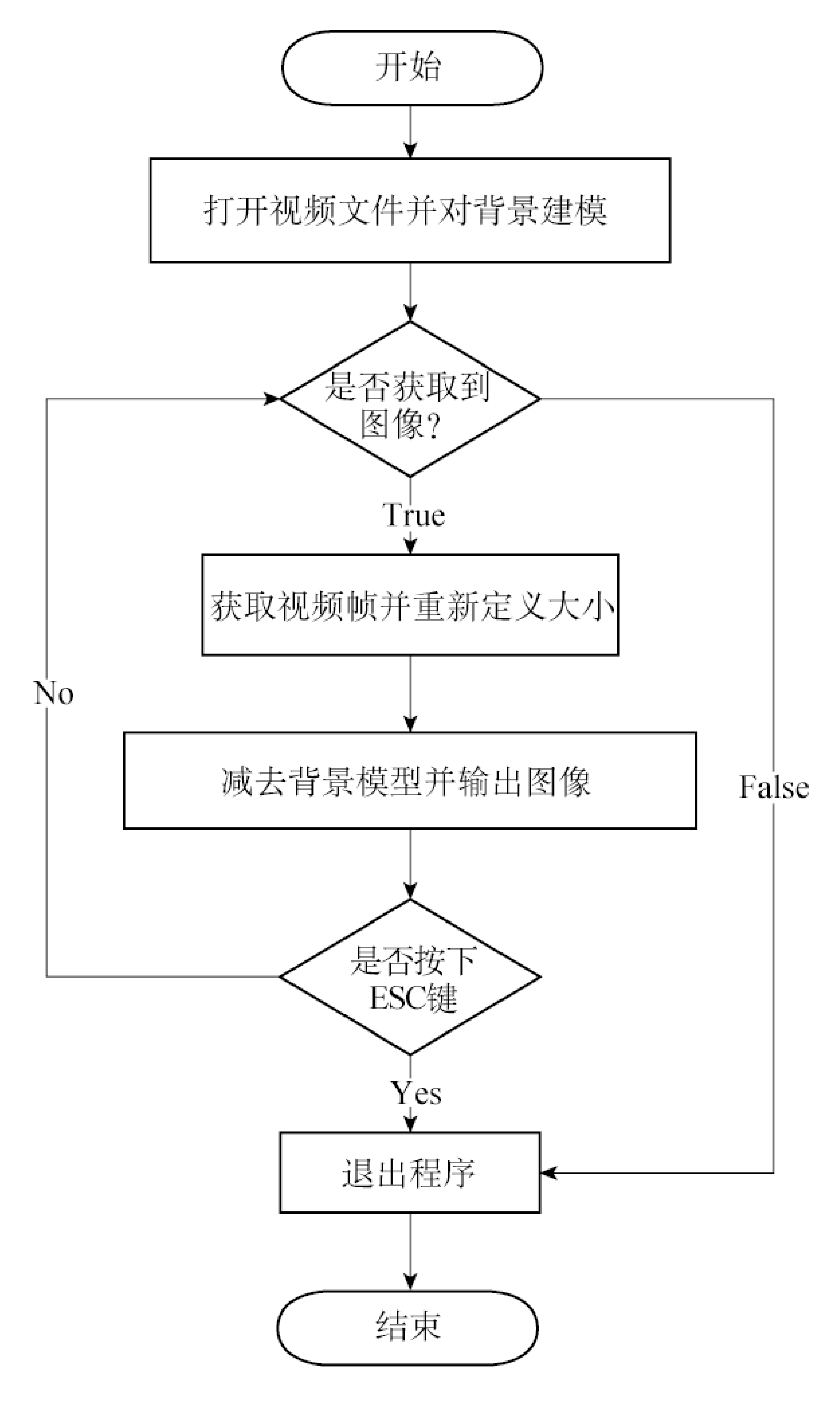


图 17.8　背景差分法跟踪物体的流程图

接下来，我们对一段道路检测视频进行分析，如程序 17.4 所示为应用背景差分法进行物体追踪的结果。

程序 17.4　背景差分法跟踪物体：

```
import cv2
import numpy as np

cap=cv2.VideoCapture('road.avi')
background_sub=cv2.createBackgroundSubtractorMOG2()
history=100
learning_rate=1.0/history
while True:
    _,frame=cap.read()
    frame=cv2.resize(frame,None,fx=0.5,fy=0.5,interpolation=cv2.INTER_AREA)
    mask=background_sub.apply(frame,learningRate=learning_rate)
    mask=cv2.cvtColor(mask,cv2.COLOR_GRAY2BGR)
    cv2.imshow('Input',frame)
    cv2.imshow('Output',mask & frame)
    key=cv2.waitKey(100)
    if key==27:
        break
cv2.destroyAllWindows()
```

输出，如图 17.9 所示：

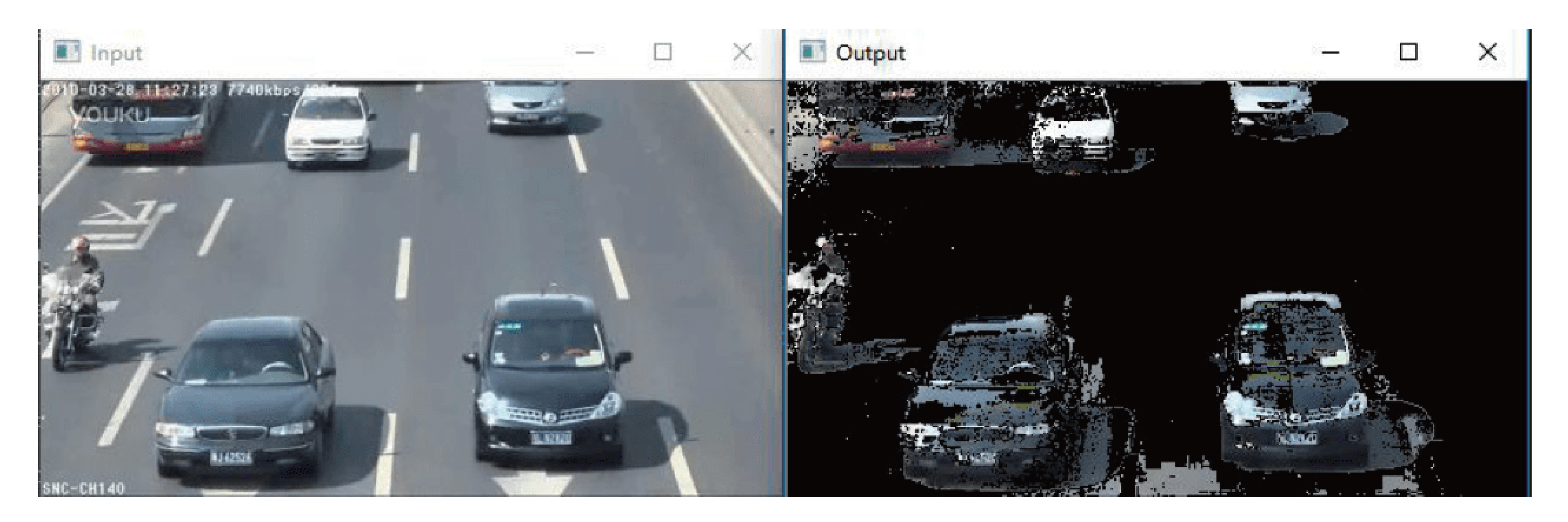


图 17.9　原视频与进行背景差分后的视频对比图

分析：

程序 17.4 应用了一个背景建模中的经典算法——混合高斯分布（GMM），OpenCV 将

GMM 封装成 createBackgroundSubtractorMOG2()，利用 createBackgroundSubtractorMOG2()完成了背景建模。定义了一个 history 参数，代表用来训练背景的帧的数目，history 参数与学习率 learning_rate 相互影响，history 越大，学习率越低。在重新定义好捕捉到帧的大小后，调用 apply()函数来进行运动检测，得到的掩码 mask 是灰度图，再转化成 RBG 图，此时 cvtColor()函数的参数是 cv2.COLOR_GRAY2BGR。

在这个程序中，可以看到，公路以及路两侧的地砖被模型化，排除掉背景干扰，我们可以比较清楚地捕获到道路上飞快移动的车辆，以便后面更深程度地对视频进行处理，如计算车速是否超速等。

17.5　利用颜色空间进行物体跟踪

假定要设计一个系统，对网球比赛进行记分，在这种场景下，我们需要捕捉球的轨迹，是否过网、是否出界等，而此时背景差分法似乎就不那么好用了，因为在球场上除了运动的球还有运动员，我们不需要捕捉运动员的轨迹。这时候，我们考虑是否可以利用颜色进行物体跟踪呢？限定网球的颜色范围，跟踪视频中符合颜色范围的物体，这是一种利用颜色空间进行目标跟踪的方法。

还是以道路监控视频为例，这次我们利用颜色空间来对物体进行跟踪，看看会有什么不同，具体程序如程序 17.5 所示。

程序 17.5　利用 HSV 颜色模型跟踪物体：

```
import cv2
import numpy as np

cap=cv2.VideoCapture(' road.avi')
while True:
    _,frame=cap.read()
    frame=cv2.resize(frame,None,fx=0.75,fy=0.75, interpolation=cv2.INTER_AREA)
    hsv=cv2.cvtColor(frame,cv2.COLOR_RGB2HSV)
    lower=np.array([0,40,40])
    upper=np.array([150,255,255])
    mask=cv2.inRange(hsv,lower,upper)
    res=cv2.bitwise_and(frame,frame,mask=mask)
    img=cv2.medianBlur(res,5)
```

```
    cv2.imshow('Input',frame
    cv2.imshow('Ouput',img)

    key=cv2.waitKey(100)
    if key==27:
      break
cv2.destroyAllWindows()
```

输出，如图 17.10 所示：



图 17.10　原视频与“筛选”特定颜色后的视频对比图

分析：

在程序 17.5 中，cvtColor()函数的参数变成了 cv2.COLOR_RBG2HSV，是将 RGB 图像转化成 HSV 形式。长久以来，RBG 颜色模型的应用最为广泛也广为人知，但是在物体追踪中并不能很好地得到我们想要的结果，所以在物体追踪中我们通常使用 HSV 颜色模型，这是一种最接近人眼对颜色的感知的模型。HSV 颜色模型分别代表 H（色调）、S（饱和度）、V（明度）。

程序将图像转化成 HSV 颜色模型的图像后，设定两组数值对图像的颜色范围进行限定，得到图像中的感兴趣区，即为 mask（掩膜），将感兴趣区外的图像值设置为 0。后面我们使用了一个中值滤波函数 medianBlur()，消除图像噪声，使图像边缘平滑，这个函数有三个参数，分别是输入图像、输出图像以及滤波模板的尺寸大小，这个尺寸必须是大于 1 的奇数。

在程序 17.5 中的道路监控视频示例中，可以看到如图 17.10 所示的输出图像。左侧为原视频图像，右侧为根据所限定的颜色范围进行物体追踪后的图像。

注意：

我们可以看到，在这个例子中利用颜色空间进行物体跟踪，由于颜色范围界定的原因，

导致对于目标物体的捕捉并不十分精准，受到光影、背景等诸多因素的影响。

可以设想最开始我们举的网球比赛记分系统的例子，将颜色限定为网球的黄色，使用颜色空间来对网球进行捕捉是否可以准确获得我们想要的结果呢？运动员的衣服颜色会不会对结果造成影响？颜色限定在黄色那么如何对网球场地的线进行捕捉呢？如果让你设计一个网球比赛的记分系统，你能想到什么样的办法呢？请同学们自行讨论。

这两节我们学习了物体跟踪的两种方法，除此之外还有帧差分法、CAMShift 算法等方法可以对物体进行跟踪。计算机视觉的应用不仅仅是物体探测和运动跟踪，还有模式识别、图像理解等方面，这是一个范围十分广的领域，能够发展的空间也很大，这里我们只挑选了几个比较简单的方面进行了介绍，如果你对计算机视觉方向很感兴趣，可以考虑进行更深层次的学习研究。

17.6　人脸识别技术

最后，我们来了解一下生活中最常见的计算机视觉应用——人脸识别系统。现今，人脸识别已经应用在我国各大企业、单位、机场等场所的身份认证流程。人脸识别系统，顾名思义就是采集人脸图像并与数据库中已经采集过的人脸图像进行比对，从而进行身份验证的系统。电视节目《最强大脑》中曾经有过这样一场特殊的比赛，百度公司旗下的人工智能机器人“小度”与“最强大脑”王峰对近千张人物的小学照片和成年照片进行比对，最后人工智能以 3∶2 获胜。如果说，人对于脸的记忆是基于大致轮廓和整体气质，那么机器人就是通过算法和学习将这些一一分解，这个过程需要依靠大规模的神经网络，万亿级别的参数，千亿的训练数据，亿级别的特征以及一些推理能力，是非常强大的人工智能系统。

人脸识别系统的工作流程如图 17.11 所示。

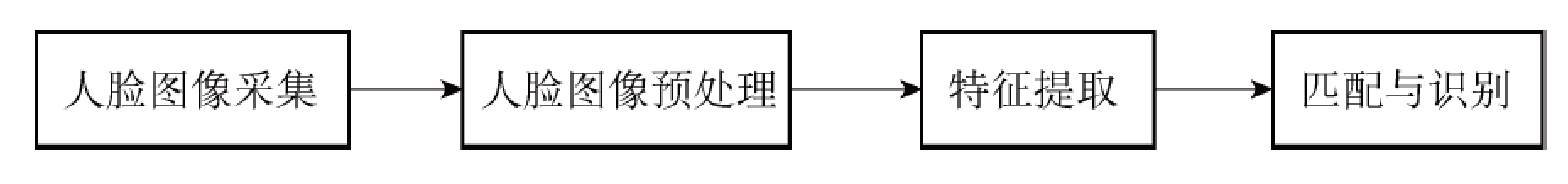


图 17.11　人脸识别系统的大致工作流程

人脸图像采集是通过摄像机对进行信息采集的人员进行拍摄，形成面部图像文件，保存在数据库中；预处理是因为在不同环境下对人脸的采集受到光照等诸多外部因素的影响，不利用后续的特征提取以及与数据库进行比对，所以要对采集到的人脸图像进行预处理，

这个预处理操作就是我们之前简单学习过的灰度变化、直方图均衡化，还有光线补偿、归一化等操作；特征提取是基于人的面部器官进行特征建模，如五官之间的距离、各个特征之间的角度关系等；最后在进行人脸识别时，通过对摄像头采集到的人脸特征模型与数据库中已经采集过并保存的数据进行对比，确定人员身份。考虑到人脸识别系统整体程序的复杂性，我们用一个实例，学习一下人脸识别过程中的第一个小步骤——人脸检测。

在进行人脸识别的时候，人物距离摄像头的远近、外界的环境均有可能对结果造成影响，所以我们需要先在图像中检测到哪一部分是人脸，这个过程就叫做人脸检测，如程序 17.6 所示。

程序 17.6　人脸检测：

```
import cv2
face_cascade = cv2.CascadeClassifier(r'haarcascade_frontalface_default.xml')
image = cv2.imread('photo.jpg')
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray,scaleFactor = 1.15,minNeighbors = 5,
minSize = (5,5),flags = cv2.CASCADE_SCALE_IMAGE)
for(x,y,w,h) in faces:
    cv2.rectangle(image,(x,y),(x+w,y+w),(0,255,0),2)
    cv2.imshow("Find Faces!",image)
    cv2.waitKey(0)
```

输出，如图 17.12 和图 17.13 所示：



图 17.12　原图

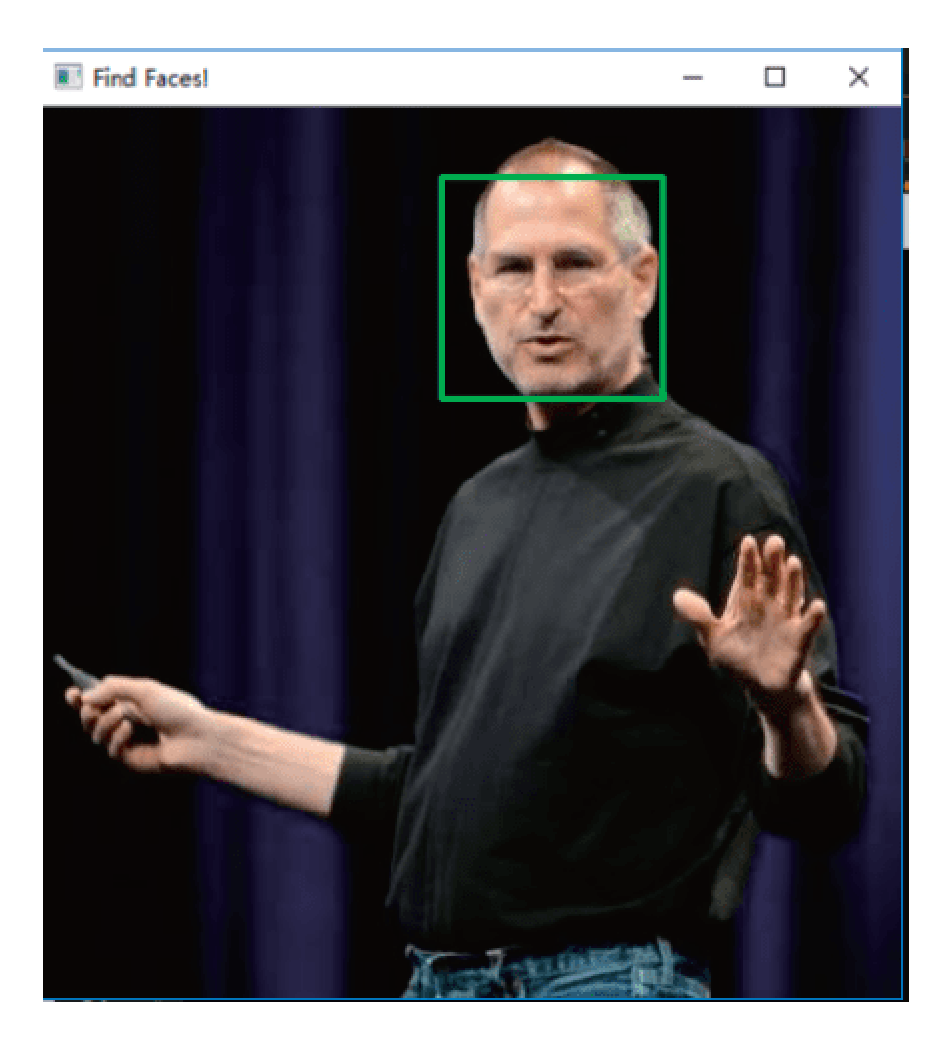


图 17.13　检测到人脸的图像

分析：

在程序 17.6 中，我们使用了 CascadeClassifier()函数，这是一种级联分类器，这里我们选用的是级联分类器中的'haarcascade_frontalface_default'，这个 XML 文件中包含有大量人脸的特征值，包括眼睛、鼻子、嘴、眉毛等特征。

后面我们调用的 detectMultiScale()函数可以检测出照片中的人脸，这个函数有多个参数，我们需要注意的是第五个参数 flags，这里 flags 参数的值设定为 CASCADE_SCALE_IMAGE，代表函数的输出形式是 rect 型，除此之外还有 CASCADE_DO_ROUGH_SEARCH，输出的是 vector 型。而 rectangle()函数则是如它的函数名一样，画一个矩形，将识别出来的人脸部分圈起来。

注意：

人脸识别系统的过程看起来很简单，但里面包含了很多算法、模型，这里我们就不一一讲解了，感兴趣的同学在学习完这一部分后不妨自己动手试一试。

这样我们就大致了解了人脸识别系统的工作过程，实际上这个技术涉及的知识和算法远远比我们介绍的要多得多，需要考虑到的情况也更加复杂，如胖瘦变化、衰老变化、化妆、头发的遮挡等因素均应考虑在内。所以说这不是一个一旦设计好就可以弃之不管的系统，它还有很大的发展空间，也随着技术的不断深入，有着很多可以优化的环节。计算机技术就是这样富有魅力，永无止境，永无极限！

从 2017 年苹果公司旗下的 iPhone X 手机上市起，利用人脸识别技术进行手机解锁已经成为智能手机发展的一个新的台阶，OPPO、HUAWEI 等多家公司随后也都发布了人脸识别手机解锁的新产品，“刷脸”时代的开始也正式代表了人脸识别技术即将进入全面普及。人脸识别技术的应用领域远比我们想象中的多，除了基本的人员身份认证，还可以应用在公安系统刑侦方面追捕罪犯，应用在银行系统加大信息安保力度，应用在电子商务的交易过程中等。

人脸识别系统是计算机视觉领域的一个小成果，计算机视觉的发展与应用还远不止于此，我国人工智能正处于一个高速发展的时代，“人才是第一资源，创新是第一动力”，未来科技会发展成什么样子没有人可以预知，就像十年前“钢铁侠”是科幻，而现在，这是一种科技。

【新闻】人脸识别技术已经发展得日益成熟，大量的数据显示人脸识别的准确率已经

达到 99.5%。近日，多伦多大学教授 Parham Aarabi 和他的研究生 Avishek Bose 开发了一种算法，可以干扰人脸识别系统从而达到保护用户个人隐私的目的。反人脸识别系统通过在图像识别过程中加入一点点干扰信息，可以将识别的成功率降低到 0.5%。这使用了一种对抗技术，让两个神经网络进行对抗，一个神经网络从数据中获取人脸数据信息，另一个神经网络试图去破坏第一个神经网络执行的任务。

人脸识别技术为人们提供了很大的便利，同时不可避免地存在安全漏洞，例如，当我们在社交网络上，上传我们的照片，那么不良商家就可以利用人脸识别系统对用户的个人信息进行窃取。那么，人脸识别技术和反人脸识别技术，究竟哪一个对我们的生活更加有益呢？我们是否应该为了保护个人信息而弃用人脸识别系统呢？请同学们自行讨论。

17.7　总　　结

本章我们学习了人工智能的一个分支领域——计算机视觉，学习了利用 PIL 库对图像进行基本处理和操作，还学习了一个功能十分强大的计算机视觉类库 OpenCV 以及一些对视频的基本操作方法，了解了两种对运动物体进行跟踪的方法，最后我们简单学习了人脸识别技术，并学习了人脸检测的方法。

通过本章的学习，我们基本了解了计算机视觉的基础知识和简单应用，当然这只是计算机视觉领域的冰山一角，像是现在比较热门的无人驾驶汽车、无人机项目，还有医学影像方面的应用均离不开计算机视觉，要想更深入更系统地学习计算机视觉，需要我们进入大学后继续研究。

17.8　练　　习

（1）任选两幅颜色接近但不相同的图片，将这两幅图片的尺寸统一后画出它们的直方图，并分析两者的区别。

（2）利用背景差分法和颜色空间法对一段网球比赛视频进行物体跟踪，并分析这两种方法的优劣。

（3）之前我们学习了人脸检测的方法，请尝试在检测出人脸的前提下，检测出人的五官。

披荆斩棘（选做）：

设计一个“石头、剪刀、布”的游戏，游戏双方分别是计算机和用户，计算机随机产生一个手势，用户则通过调用摄像头对镜头前人的手势进行捕捉，然后进行输赢的判定。

第 18 章　人工神经网络

本章将学习人工神经网络。我们将介绍如何建立人工神经网络并且去训练它。此外，还将讨论感知器以及如何基于此构建一个分类器。我们将学习单层神经网络和多层神经网络。同时将了解循环神经网络。最后将使用人工神经网络来构建一个光学字符识别引擎。

学完本章后，将会了解：

- 什么是人工神经网络。
- 建立人工神经网络。
- 训练人工神经网络。
- 感知器。
- 单层神经网络。
- 多层神经网络。
- 循环神经网络
- 在光学字符识别（OCR）数据库中可视化字符。
- 构建一个光学字符识别（OCR）引擎。

18.1　什么是人工神经网络

人工神经网络（Artificial Neural Network，ANN），是 20 世纪 80 年代以来人工智能领域兴起的研究热点。它从信息处理角度对人脑神经元网络进行抽象，建立某种简单模型，按不同的连接方式组成不同的网络。在工程与学术界也常直接简称为神经网络或类神经网络（见图 18.1）。神经网络是一种运算模型，由大量的节点（或称神经元）之间相互连接构成。每个节点代表一种特定的输出函数，称为激励函数（Activation Function）。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出则依网络的连接方式、权重值和激励函数的不同而不同。而网络

自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。



图 18.1　人工神经网络

提示：

激励函数：神经网络中的每个节点接受输入值，并将输入值传递给下一层，输入节点会将输入属性值直接传递给下一层（隐层或输出层）。在神经网络中，隐层和输出层节点的输入和输出之间具有函数关系，这个函数称为激励函数。

人工神经网络具有自学习功能。例如，实现图像识别时，只要先把许多不同的图像样板和对应的应识别的结果输入人工神经网络，网络就会通过自学习功能，慢慢学会识别类似的图像。自学习功能对于预测有特别重要的意义。人工神经网络还具有联想存储功能。用人工神经网络的反馈网络就可以实现这种联想。此外，人工神经网络具有高速寻找优化解的能力。寻找一个复杂问题的优化解，往往需要很大的计算量，利用一个针对某问题而设计的反馈型人工神经网络，发挥计算机的高速运算能力，可以很快找到优化解。

最近十多年来，人工神经网络的研究工作不断深入，已经取得了很大的进展，其在模式识别、智能机器人、自动控制、预测估计、生物、医学、经济等领域已成功地解决了许多现代计算机难以解决的实际问题，表现出了良好的智能特性。

18.2　建立人工神经网络

人工智能的一个基本前提是制造能够执行满足人类智能需求的任务的机器。人类的大

脑在学习新事物方面是惊人的。为什么不使用人类大脑的模型来制造一台机器呢？人工神经网络是一种模仿生物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点（神经元）之间相互连接的权重，从而达到处理信息的目的。

人工神经网络是由大量处理单元经广泛互连而组成的人工网络，用来模拟脑神经系统的结构和功能。而这些处理单元我们把它称作人工神经元。人工神经网络可看成是以人工神经元为节点，用有向加权弧连接起来的有向图。有向弧的权值表示相互连接的两个人工神经元间相互作用的强弱。人工神经元结构如图 18.2 所示。

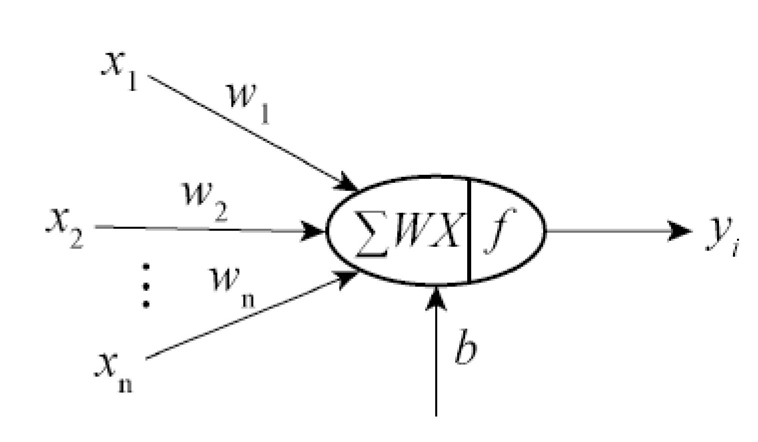


图 18.2　人工神经元模型

人工神经网络的设计使它们能够识别数据中的潜在模式并从中学习。它们可以用于各种任务，如分类、回归、分割等。我们需要将任何给定的数据转换成数字形式，然后再将其输入到神经网络中。例如，我们处理许多不同类型的数据，包括可视的、文本的、时间序列等。我们需要弄清楚如何用一种可以被人工神经网络理解的方式来表示问题。

人类的学习过程是分等级的。我们大脑的神经网络有不同的阶段，每个阶段都对应着不同的粒度。有些阶段学习简单的东西，有些阶段学习复杂一些的东西。让我们考虑一个视觉识别物体的例子。当我们看一个盒子时，第一阶段会识别出一些简单的东西，如角和边缘。下一阶段识别了通用的形状，之后又确定了它是什么类型的物体。这个过程对于不同的任务是有差异的。通过构建这个层次结构，我们的大脑快速地分离了概念并识别出了给定的物体，如图 18.3 所示。

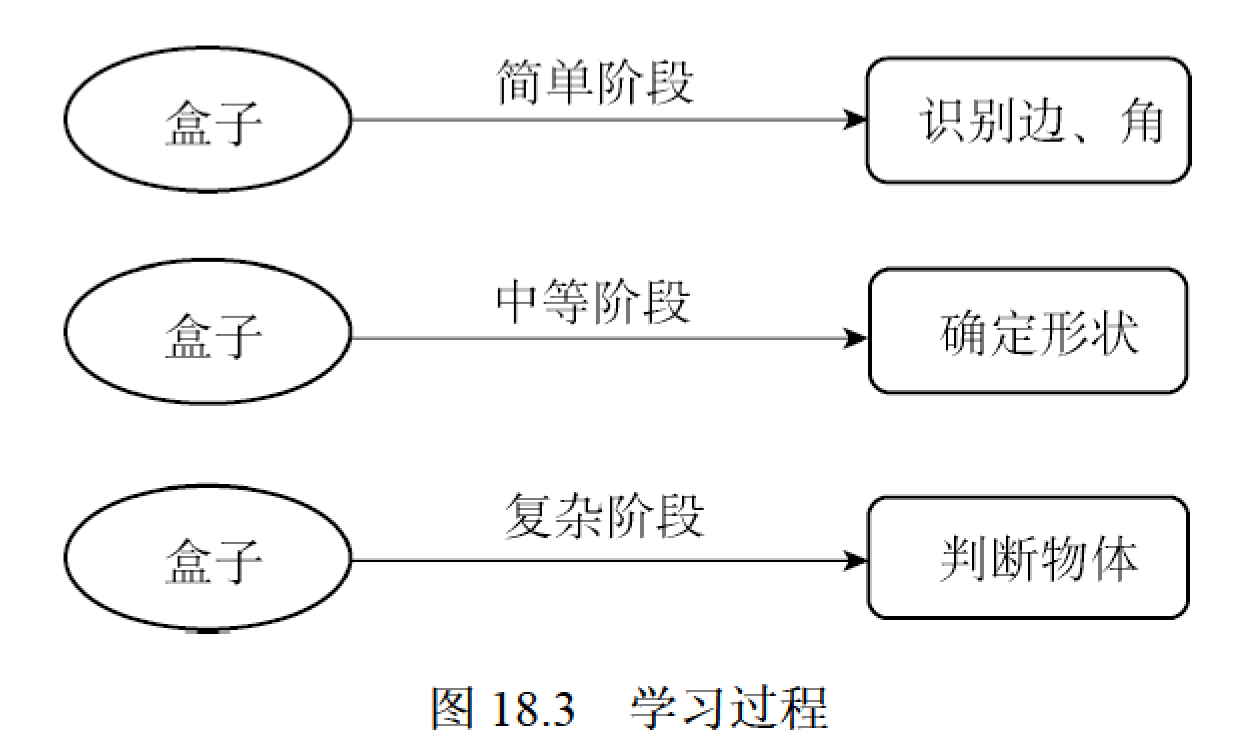


图 18.3　学习过程

为了模拟人类大脑的学习过程，人工神经网络是用神经元层构建的。这些神经元的灵感来自于生物神经元。人工神经网络中的每一层都是一组独立的神经元。一层中的每个神经元与相邻层的神经元相连。人工神经网络结构示意图如图 18.4 所示。

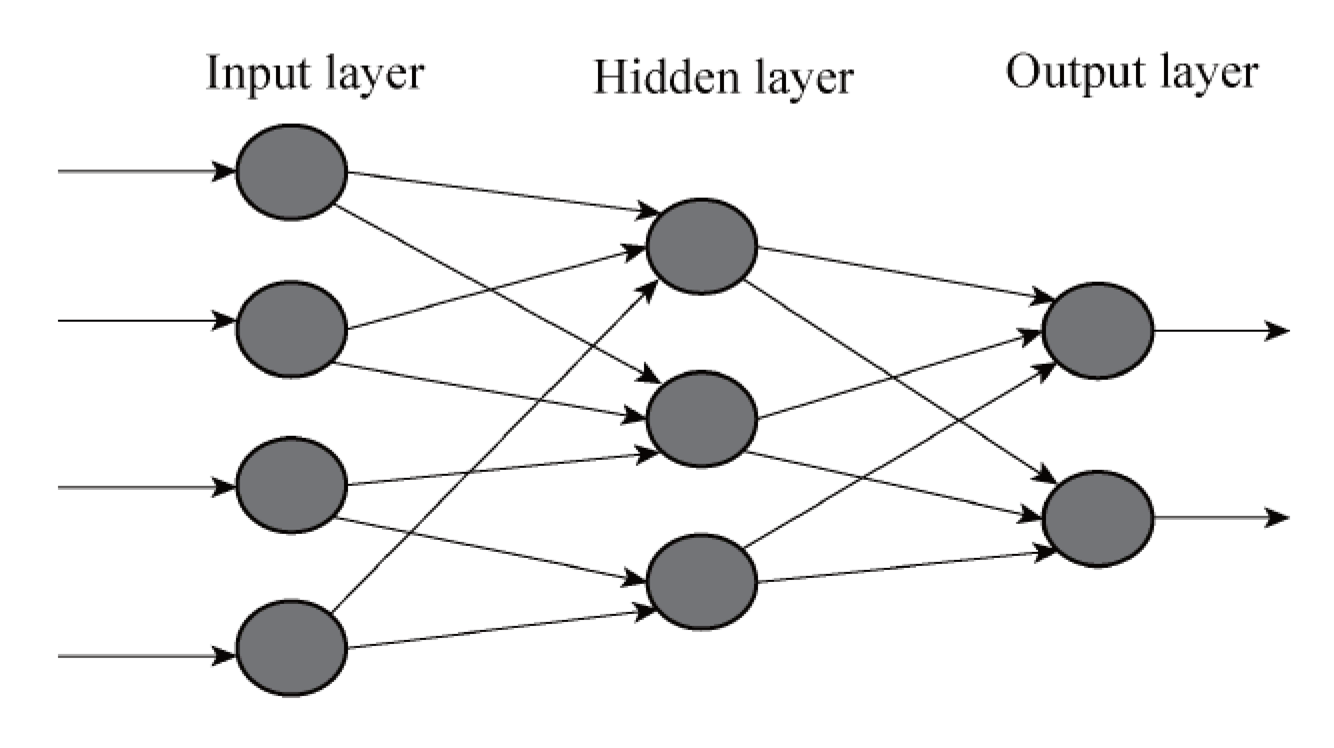


图 18.4　人工神经网络示意图

人工神经网络中，神经元处理单元可表示不同的对象，如特征、字母、概念，或者一些有意义的抽象模式。网络中处理单元的类型分为三类：输入单元、输出单元和隐单元。输入单元接受外部世界的信号与数据；输出单元实现系统处理结果的输出；隐单元是处在输入和输出单元之间，不能由系统外部观察的单元。神经元间的连接权值反映了单元间的连接强度，信息的表示和处理体现在网络处理单元的连接关系中。人工神经网络是一种非程序化、适应性、大脑风格的信息处理，其本质是通过网络的变换和动力学行为得到一种并行分布式的信息处理功能，并在不同程度和层次上模仿人脑神经系统的信息处理功能。

18.3　训练人工神经网络

如果我们处理的是 N 维输入数据，那么输入层将由 N 个神经元组成。如果我们的训练数据中有 M 个不同的类，那么输出层将由 M 个神经元组成。输入和输出层之间的层称为隐藏层。一个简单神经网络将由几个层组成，一个深度神经网络将由许多层组成，如图 18.5 所示。

考虑一个我们想要使用神经网络对给定数据进行分类的情况。第一步是收集适当的训练数据并对其进行标记。每个神经元都是一个简单函数，神经网络会自动训练，直到误差

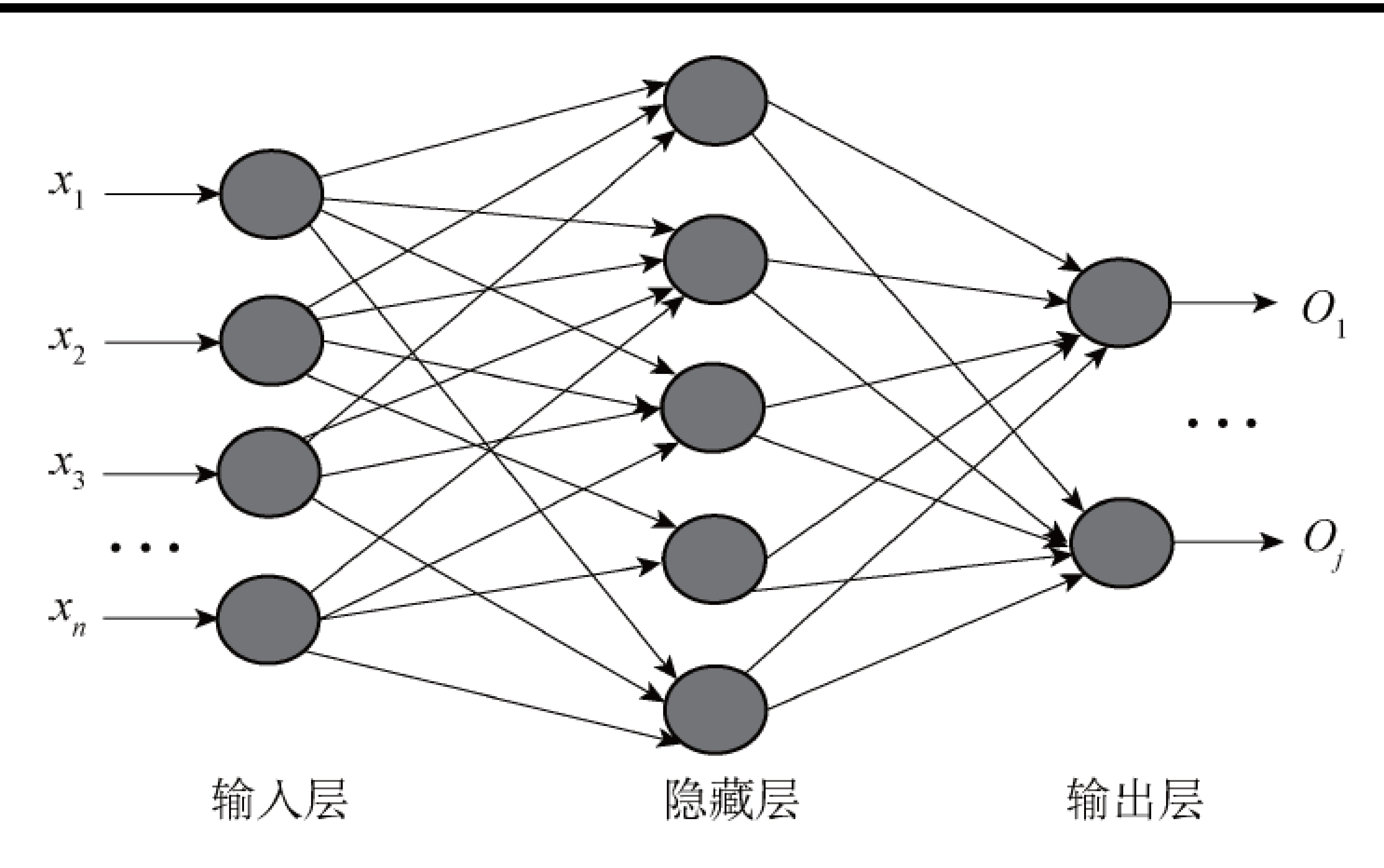


图 18.5　人工神经网络结构图

低于某个值。误差基本上是预测输出和实际输出之间的差值。根据误差有多大，神经网络会自我调整，重新训练，直到它接近解决方案。

人工神经网络具有自学习和自适应的能力，可以通过预先提供的一批相互对应的输入和输出数据，分析两者的内在关系和规律，最终通过这些规律形成一个复杂的非线性系统函数，这种学习分析过程被称作“训练”。神经元的每一个输入连接都有连接强度，用一个连接权值来表示，即将产生的信号通过连接强度放大，每一个输入量都对应有一个相关联的权重。处理单元将经过权重的输入量化，然后相加求得加权值之和，计算出输出量，这个输出量是权重和的函数，一般称此函数为传递函数。

18.4　感　知　器

感知器，也可翻译为感知机，是 Frank Rosenblatt（弗兰克 • 罗森布拉特）在 1957 年就职于 Cornell 航空实验室（Cornell Aeronautical Laboratory）时所发明的一种人工神经网络。它可以被视为一种最简单形式的前馈式人工神经网络（下一节介绍），是一种二元线性分类器。它是人工神经网络中的一种典型结构，它的主要特点是结构简单，对所能解决的问题存在着收敛算法，并能从数学上严格证明，从而对神经网络研究起到重要的推动作用。

感知器是使用特征向量来表示的前馈式人工神经网络，它是一种二元分类器，把矩阵上的输入（实数值向量）映射到输出值 $f(x)$上（一个二元的值）。

$$f(x)=\begin{cases}1 & \text{if } w \bullet x + b > 0 \\ 0 & \text{else}\end{cases}$$

w 是实数的表示权重的向量，$w \bullet x$ 是点积。b 是偏置，一个不依赖于任何输入值的常数。偏置可以认为是激励函数的偏移量，或者给神经元一个基础活跃等级。$f(x)$（0 或 1）用于进行分类，看它是肯定的还是否定的，这属于二元分类问题。如果 b 是负的，那么加权后的输入必须产生一个肯定的值并且大于 b，这样才能令分类神经元大于阈值 0。从空间上看，偏置改变了决策边界的位置（虽然不是定向的）。

感知器是人工神经网络的组成部分，如图 18.6 所示。它是一个单神经元，它接受输入，对它们进行计算，然后产生输出。它使用一个简单线性函数来做决定。假设我们处理的是 N 维输入数据点。感知器计算这些 N 个数字的加权总和，然后它添加一个常数来产生输出。这个常数被称为神经元的偏置。值得注意的是，这些简单的感知器经常被用来设计非常复杂的深度神经网络。

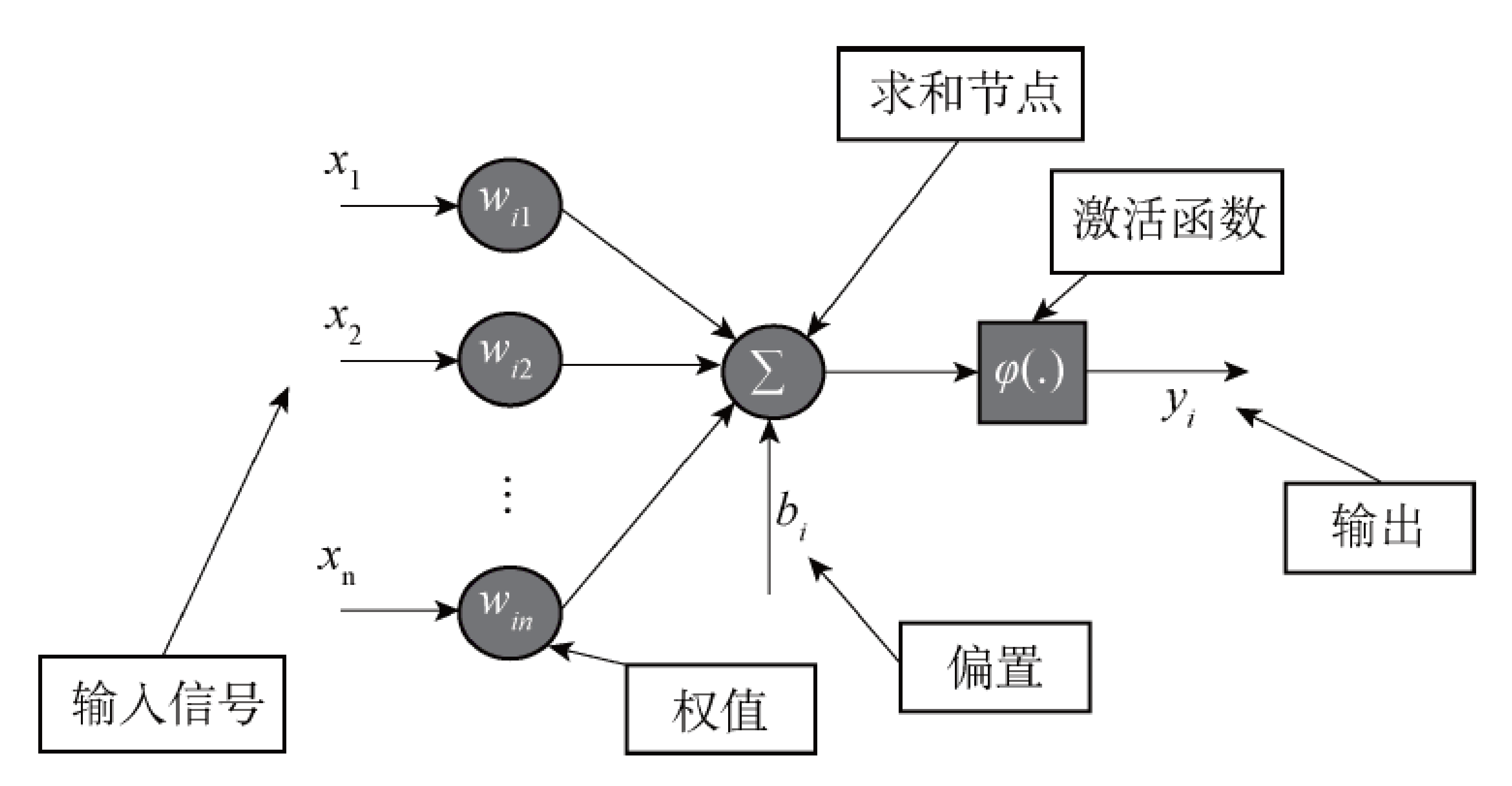


图 18.6　感知器示意图

在人工神经网络领域中，感知器也被指为单层的人工神经网络，以区别于较复杂的多层感知器（Multilayer Perceptron）。作为一种线性分类器，（单层）感知器可说是最简单的前向人工神经网络形式。尽管结构简单，但感知器能够学习并解决相当复杂的问题。感知器主要的本质缺陷是它不能处理线性不可分问题。

下面我们通过程序 18.1 来加深对感知器的了解。

程序 18.1　构建感知器网络：

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import neurolab as nl

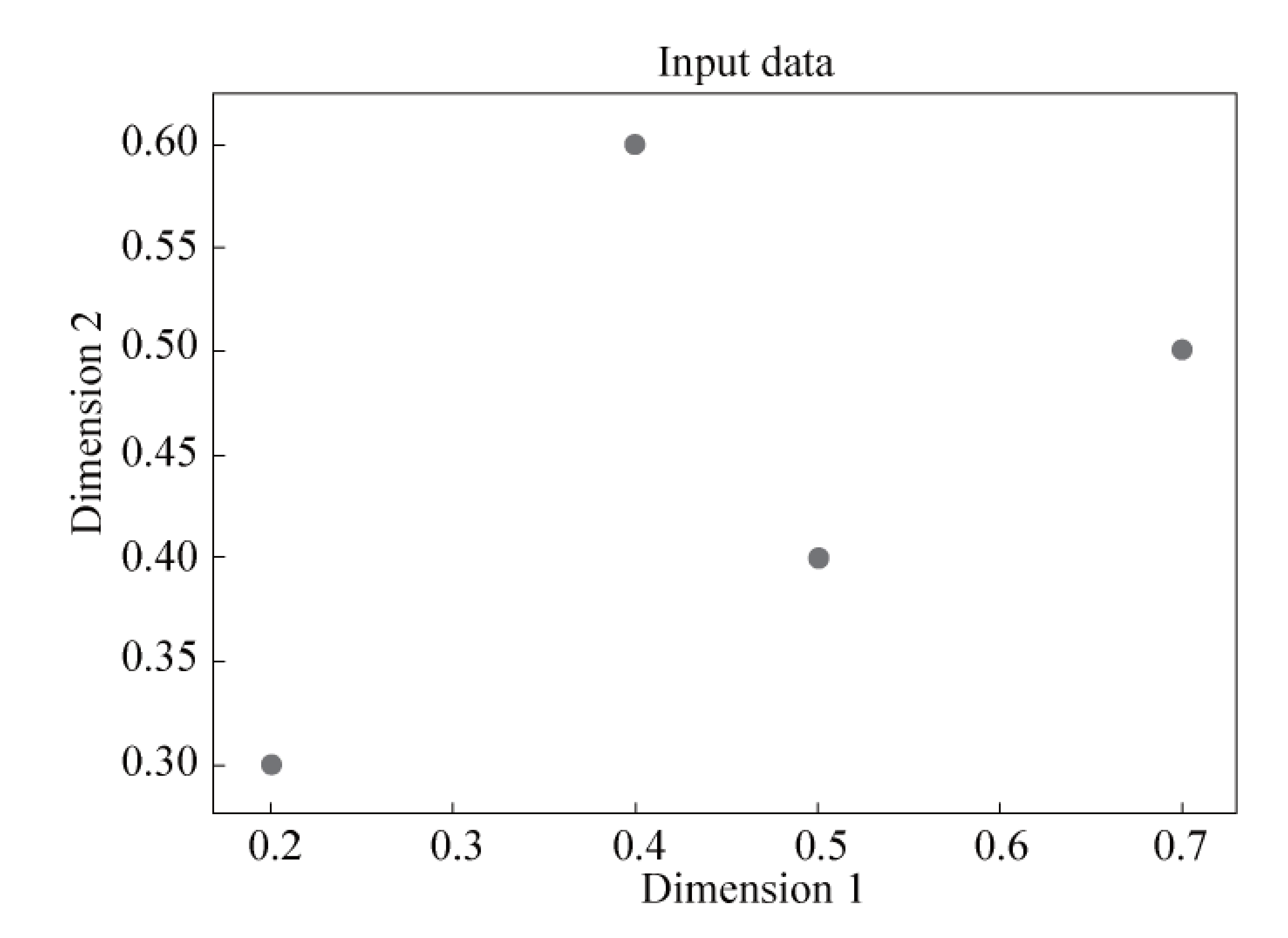
data = np.array([[0.2, 0.3], [0.5, 0.4], [0.4, 0.6], [0.7, 0.5]])
labels = [[0], [0], [0], [1]]

plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')

dim1 = [0, 1]
dim2 = [0, 1]
num_output = 1
perceptron = nl.net.newp([dim1, dim2], num_output)
error_progress = perceptron.train(data, labels, epochs=80, show=20, lr=0.03)

plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Training error progress')
plt.grid()
plt.show()
```

输出，如图 18.7 所示：



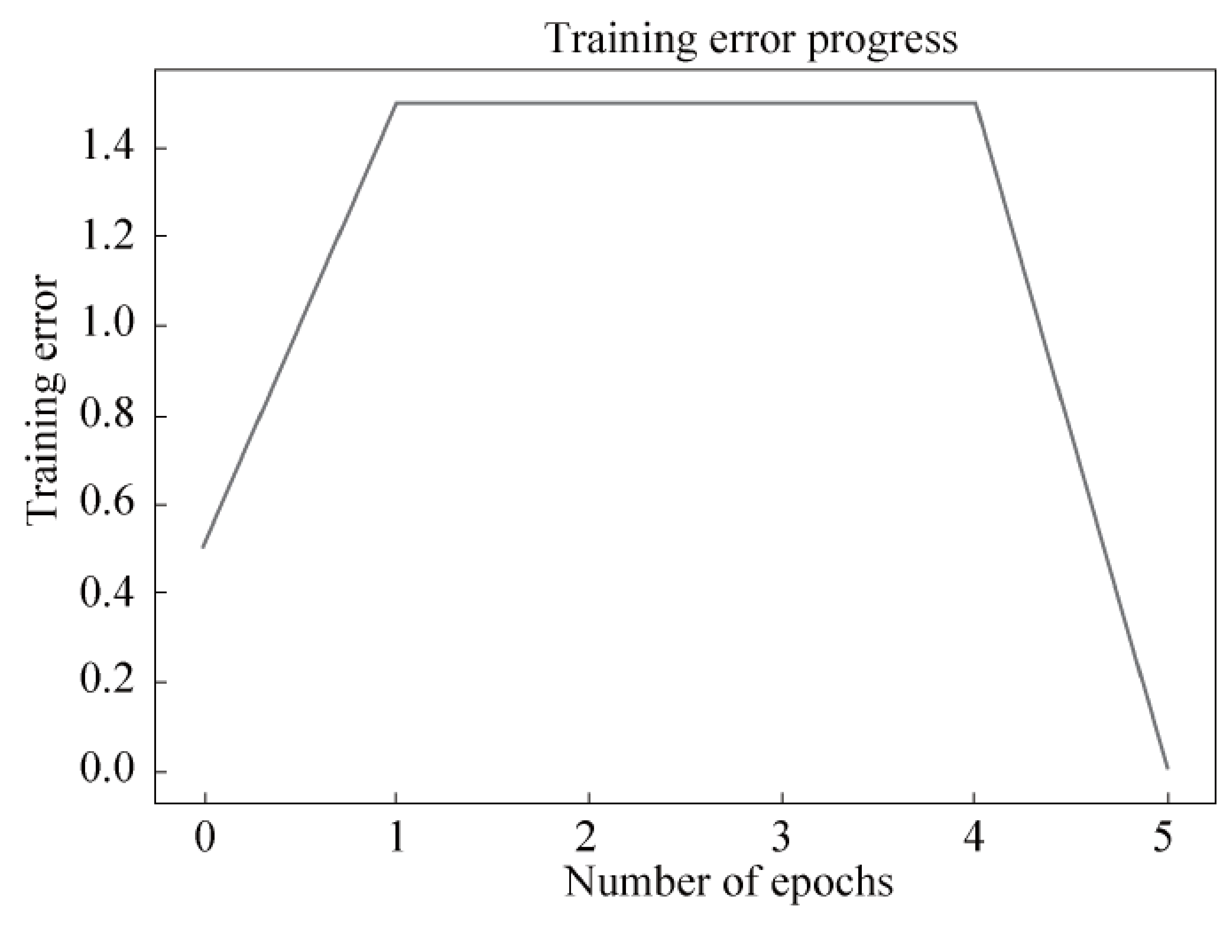


图 18.7　程序输出

分析：

本程序是构建一个基于感知器的分类器。首先，我们导入 numpy、matplotlib 和 neurolab 包。接着定义用于训练感知器的数据和标签（为什么需要标签呢？因为这是一个基于监督学习的例子，下同）。然后我们使用 matplotlib 中的 pyplot 模块（提供一个类似 matlab 的绘图框架）绘制输入数据点，程序的第 8～12 行是绘制输入数据并可视化。接下来我们定义两个向量，这个向量是二维的，它们作用于神经元的输入，再定义一个输出神经元。我们用这两个输入神经元和一个输出神经元使用 newp()方法生成一个感知器网络。再使用之前定义的数据和标签训练该感知器，train()函数中的参数 epochs 指迭代次数，show 指每迭代多少次在终端显示一次输出，lr 是学习率。最后，我们使用误差度量来绘制训练进度并可视化输出。

在程序的输出中，第一张图显示了输入的数据点，第二张图显示了使用误差度量绘制的训练进度，我们可以看到在第四个阶段的末尾，误差降为了 0。

提示：

matplotlib：它是 Python 中最常用的可视化工具之一，可以非常方便地创建海量类型的 2D 图表和一些基本的 3D 图表。matplotlib 最早是为了可视化癫痫病人的脑皮层电图相关的信号而研发，因为在函数的设计上参考了 MATLAB，所以叫作 matplotlib。

neurolab：一个简单而强大的神经网络库。包括基础神经网络、训练算法，并具有弹性的架构，可创建其他网络，它用纯 Python 和 numpy 写成。

18.5　单层神经网络

在学习单层神经网络之前，我们先学习一下神经网络模型。神经网络模型从传播来讲分为两种：前馈神经网络（前向网络）和反馈神经网络。

（1）前向网络：没有反馈机制，也就是只能向前传播而不能反向传播来调整权值参数。感知器就属于前向网络。网络中各个神经元接受前一级的输入，并输出到下一级，网络中没有反馈，可以用一个有向无环路图表示，如 18.3 节的图 18.5 就是一个前向网络。这种网络实现信号从输入空间到输出空间的变换，它的信息处理能力来自于简单非线性函数的多次复合。网络结构简单，易于实现。

（2）反馈网络：反馈型神经网络是一种从输出到输入具有反馈连接的神经网络，其结构比前馈网络要复杂得多。在这类网络中，多个神经元互连以组成一个互联神经网络。有些神经元的输出被反馈至同层或前层神经元。因此，信号能够从正向和反向流通，如图 18.8 所示。

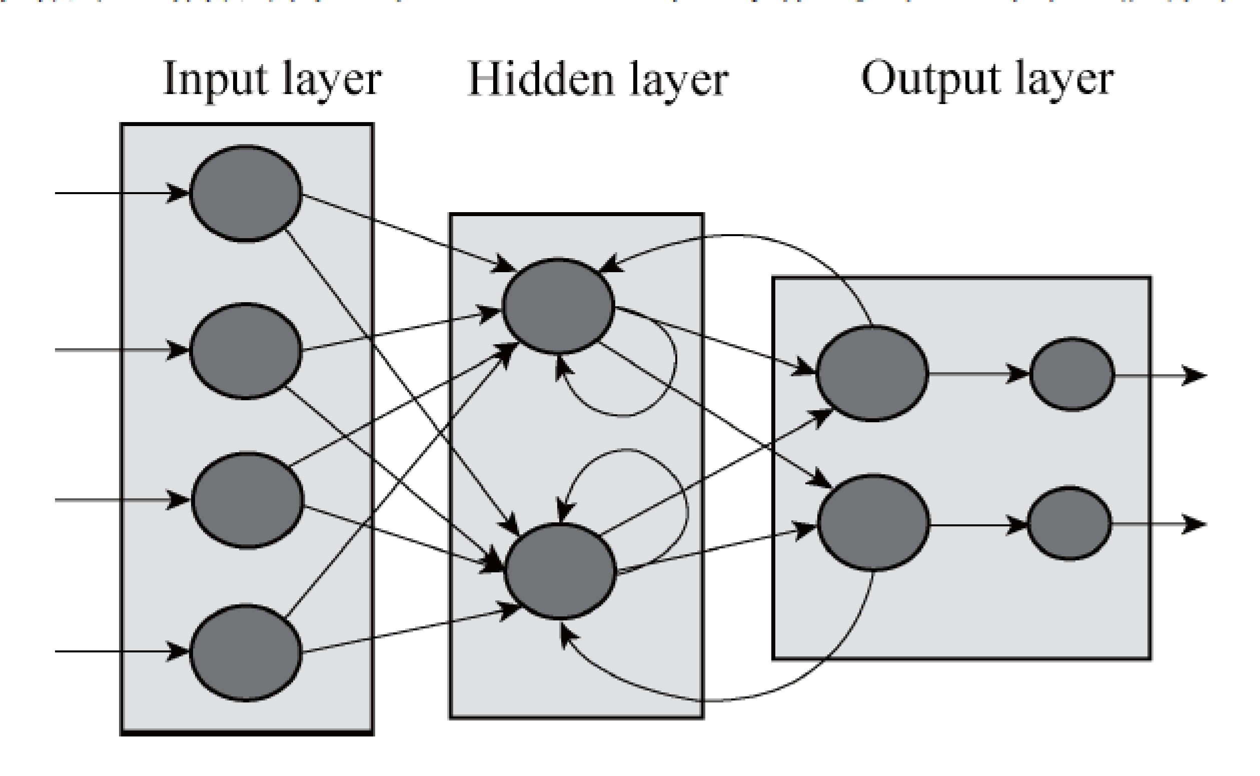


图 18.8　反馈神经网络

在反馈网络中，输入信号决定反馈系统的初始状态，然后系统经过一系列状态转换以后，逐渐收敛于平衡状态。这样的平衡状态就是反馈网络经计算后的输出结果。

单层神经网络即单层感知器（Single Layer Perceptron）。单层感知器是最简单的神经网络。它包含输入层和输出层，而输入层和输出层是直接相连的。单层神经网络一般是来

解决线性问题的，用于二分类问题，如图 18.9 所示。

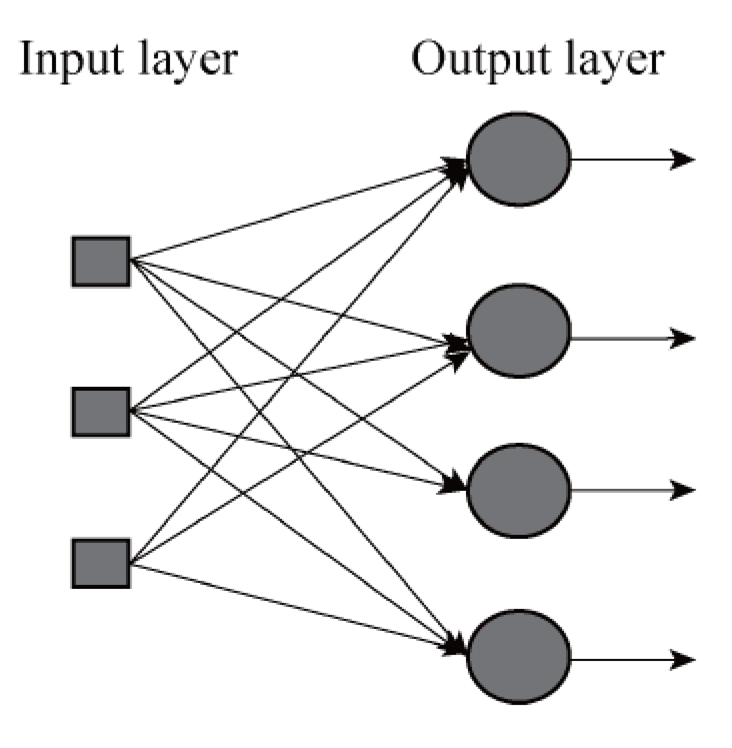


图 18.9　单层神经网络

算法思想：首先把连接权和阈值初始化为较小的非零随机数，然后把有 N 个连接权值的输入送入网络，经加权运算处理，得到的输出如果与所期望的输出有较大的差别，就对连接权值参数按照某种算法进行自动调整，经过多次反复，直到所得到的输出与所期望的输出间的差别满足要求为止。

单层神经网络不能表达的问题被称为线性不可分问题。线性不可分函数的数量随着输入变量个数的增加而快速增加，甚至远远超过了线性可分函数的个数。也就是说，单层神经网络不能表达的问题的数量远远超过了它所能表达的问题的数量。

构建单层神经网络的例子如程序 18.2 所示。

程序 18.2　构建单层神经网络：

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

text = np.loadtxt('data_simple_neural.txt')
data = text[:, 0:2]
labels = text[:, 2:]

plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')
```

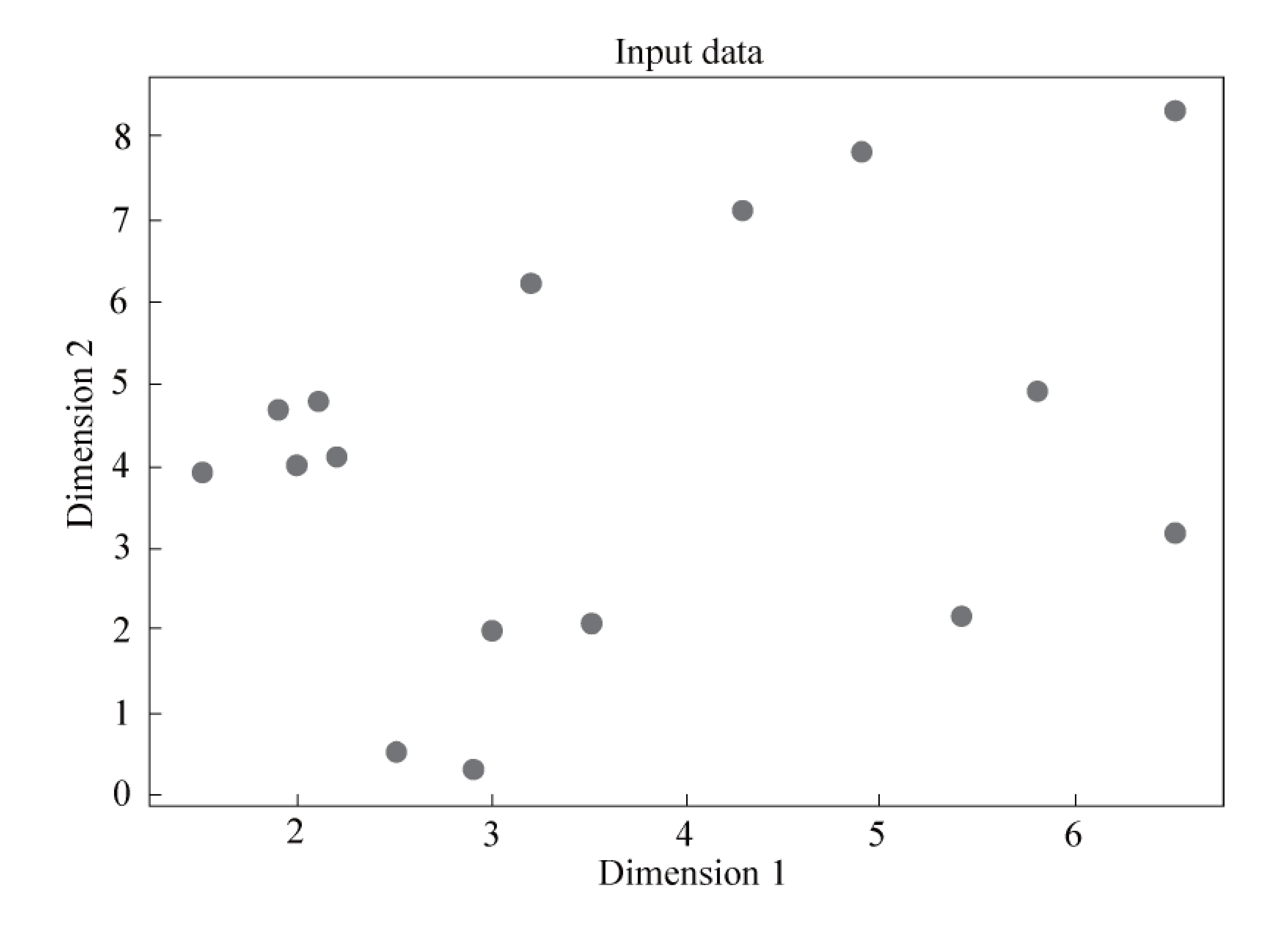
```

dim1_min, dim1_max = data[:, 0].min(), data[:, 0].max()
dim2_min, dim2_max = data[:, 1].min(), data[:, 1].max()
num_output = labels.shape[1]
dim1 = [dim1_min, dim1_max]
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)
error_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)

plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Training error progress')
plt.grid()
plt.show()

print('\nTest results:')
data_test = [[0.5, 4.2], [4.7, 0.3], [3.6, 7.9]]
for item in data_test:
    print(item, '-->', nn.sim([item])[0])
```

输出，如图 18.10 所示：



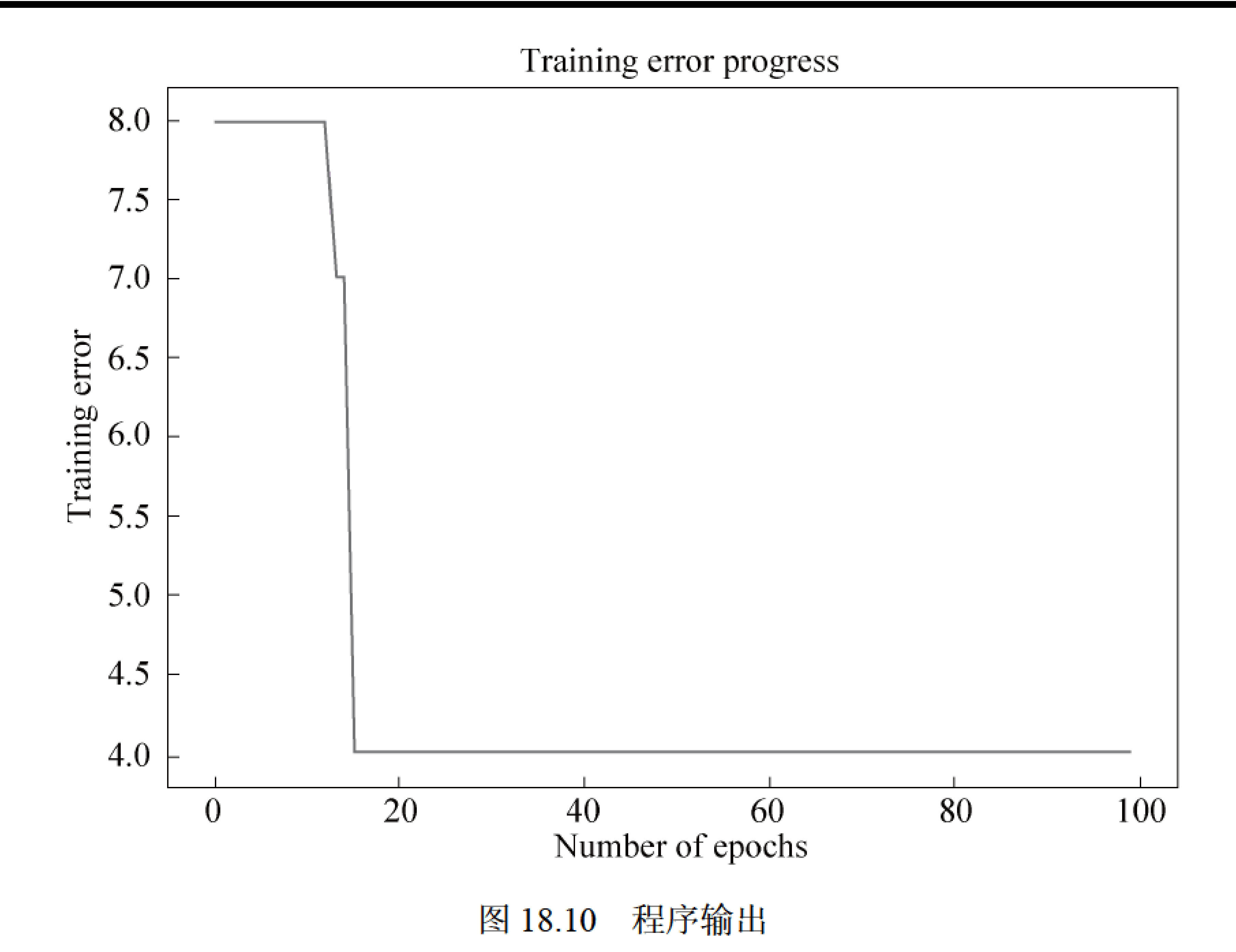


图 18.10　程序输出

```
Epoch: 20; Error: 4.0;
Epoch: 40; Error: 4.0;
Epoch: 60; Error: 4.0;
Epoch: 80; Error: 4.0;
Epoch: 100; Error: 4.0;
The maximum number of train epochs is reached

Test results:
[0.5, 4.2] --> [1. 0.]
[4.7, 0.3] --> [1. 0.]
[3.6, 7.9] --> [1. 1.]
```

分析：

程序 18.2 和程序 18.1 类似。首先导入相关包，我们使用 data_simple_neural.txt 文本中提供的数据来作为输入。该文本的每一行包含以空格作为分隔的 4 个数字，前两个数字形成了数据点，后两个数字是标签。你可以根据自己所想来定义该文本中的数据。我们在程序中加载该文本，并将输入数据分为训练数据和标签使用。接下来绘制输入数据点。以上代码完成之后，我们为每个维度提取最小值和最大值，然后定义输出层中神经元的数量。

接下来定义两个向量，向量中的数据是每个维度的最小值和最大值。我们使用这些参数生成一个单层神经网络，并使用训练数据和标签来训练该神经网络。然后，我们绘制训练进度。之后，我们定义一些测试数据点，并使用这些数据点进行网络的仿真，sim()函数实现了网络的仿真。最后打印测试结果。

在程序的输出中，第一张图显示了输入数据点，第二张图显示了训练进度。在终端的输出中显示了迭代的次数，我们规定了每迭代 20 次显示一次，迭代 100 次之后，最大迭代次数已经达到。然后是对测试数据点的测试结果，如果你在 2D 图形中查找这些测试数据点，则可以直观地验证预测的输出是否正确。

18.6　多层神经网络

为了提高精确度，我们需要给神经网络提供更多的自主性。这意味着神经网络需要不止一层来提取训练数据中的潜在模式。

MLP（Multi-layer Perceptron），即多层感知器，也就是多层神经网络。它不仅仅只有一层隐藏层，如图 18.11 所示。根据实际的情况或者是算法的需要，多层神经网络可以有两层或更多的隐藏层。它是一种前向结构的人工神经网络，映射一组输入向量到一组输出向量。MLP 可以被看作是一个有向图，由多个节点层组成，每一层全连接到下一层。除了输入节点，每个节点都是一个带有非线性激活函数的神经元（或称处理单元）。一种被称为反向传播算法的监督学习方法常被用来训练 MLP。MLP 是感知器的推广，克服了感知器不能对线性不可分数据进行识别的弱点。

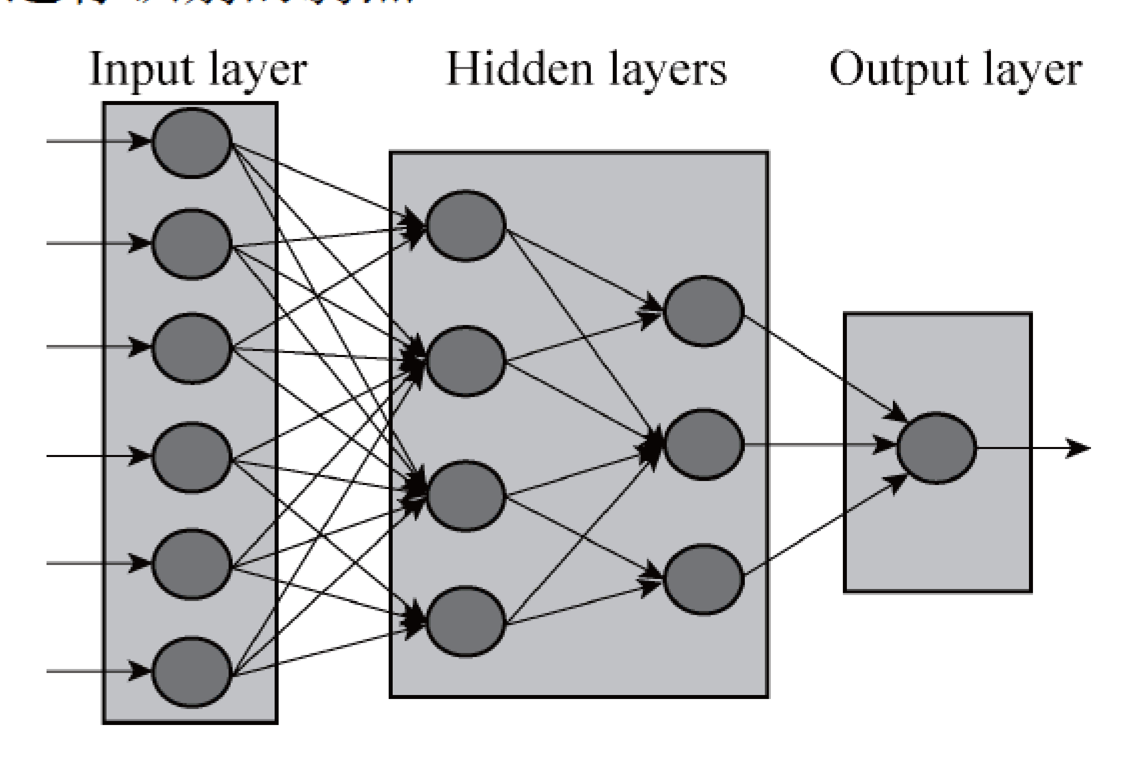


图 18.11　多层神经网络

接下来我们用代码构建一个多层神经网络的例子，如程序 18.3 所示。

程序 18.3　构建多层神经网络：

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

min_val = -20
max_val = 20
num_points = 150
x = np.linspace(min_val, max_val, num_points)
y = 2 * np.square(x) + 7
y /= np.linalg.norm(y)

data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)

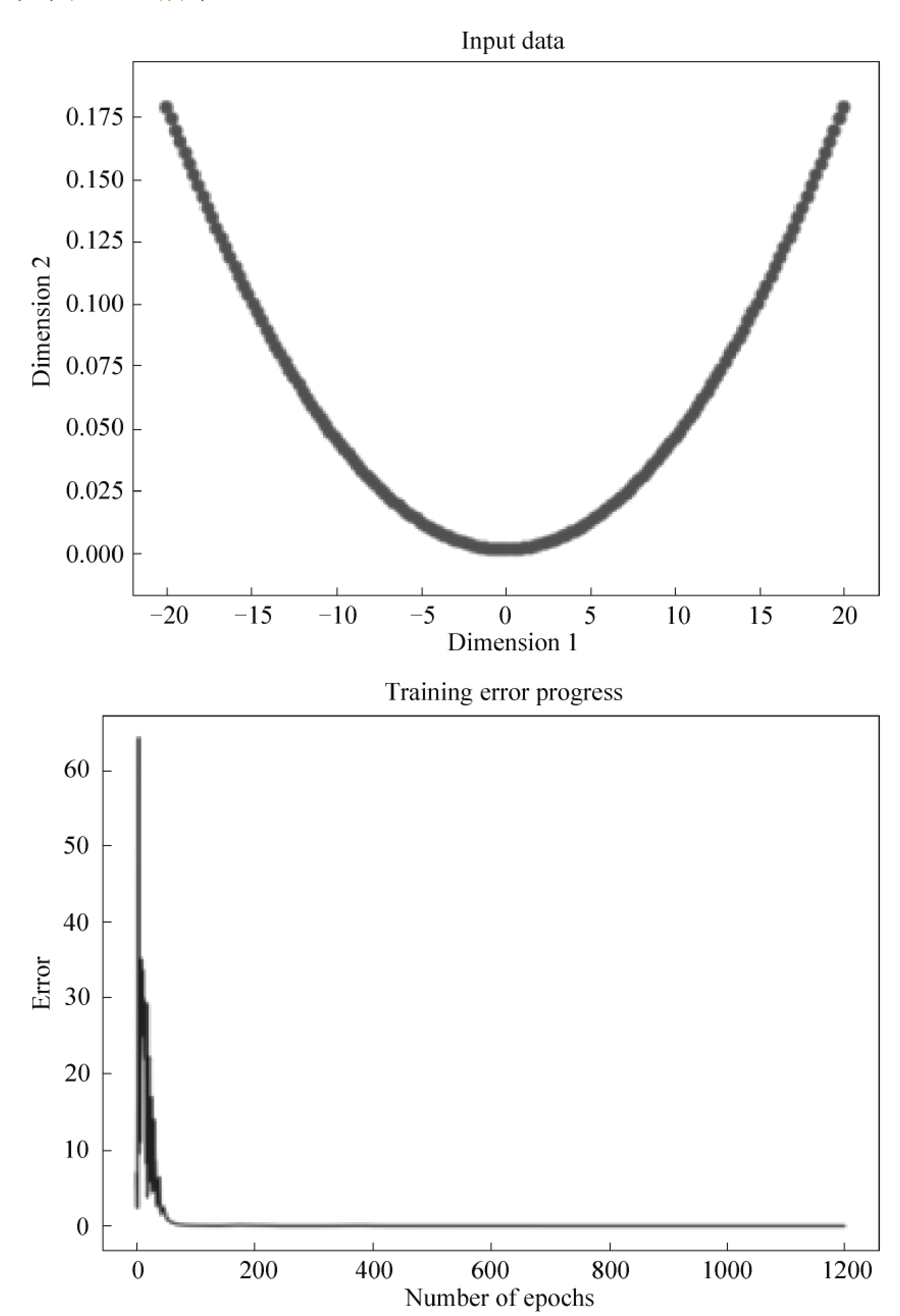
plt.figure()
plt.scatter(data, labels)
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')

nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])
nn.trainf = nl.train.train_gd
error_progress = nn.train(data, labels, epochs=1200, show=100, goal=0.01)
output = nn.sim(data)
y_pred = output.reshape(num_points)

plt.figure()
plt.plot(error_progress)
plt.xlabel('Number of epochs')
plt.ylabel('Error')
plt.title('Training error progress')

x_dense = np.linspace(min_val, max_val, num_points * 2)
y_dense_pred = nn.sim(x_dense.reshape(x_dense.size,1)).reshape(x_dense.size)
plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
plt.title('Actual vs predicted')
plt.show()
```

输出，如图 18.12 所示：


Input data
Dimension 2
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−20
−15
−10
−5
0
5
10
15
20
Dimension 1
Training error progress
Error
60
50
40
30
20
10
0
0
200
400
600
800
1000
1200
Number of epochs



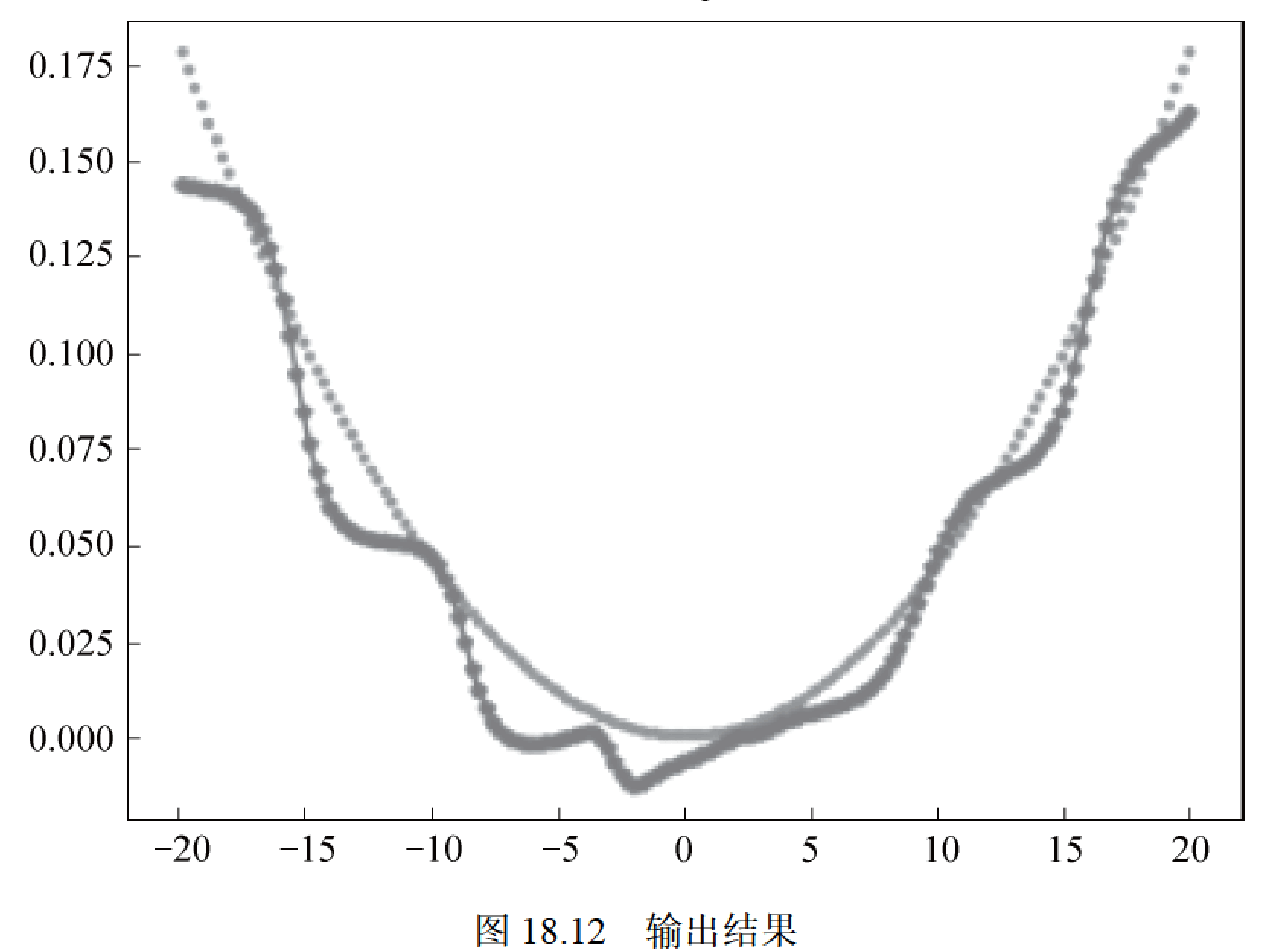


图 18.12　输出结果

```
Epoch: 100; Error: 0.11331967842182422;
Epoch: 200; Error: 0.02710494554279251;
Epoch: 300; Error: 0.08570676411013539;
Epoch: 400; Error: 0.05766512665344843;
Epoch: 500; Error: 0.05757099489196255;
Epoch: 600; Error: 0.050491014550849124;
Epoch: 700; Error: 0.04685468635504871;
Epoch: 800; Error: 0.04273185529871567;
Epoch: 900; Error: 0.03917842801570279;
Epoch: 1000; Error: 0.03589262044974862;
Epoch: 1100; Error: 0.032897717239481;
Epoch: 1200; Error: 0.030183541423230622;
The maximum number of train epochs is reached
```

分析：

这段代码的目的是创建一个多层神经网络。首先，导入相关包。接着根据方程 $y = 2x^2 + 7$ 生成一些样本数据点，然后将这些点进行规范化处理。linspace()函数的作用是在指定的间隔内返回均匀间隔的数字。linalg.norm()是将数据进行规范化处理，它是一种

数据预处理技术。重塑上述变量，创建用于训练网络的数据集和标签。接下来绘制输入数据。

完成之后，我们使用之前定义过的参数生成一个包含两个隐藏层的多层神经网络，newff()函数是生成一个前馈 BP（Back Propagation）神经网络。第一个隐藏层有 10 个神经元，第二个隐藏层有 6 个神经元。你可以自定义隐藏层中的神经元数量，也可以自定义隐藏层的数量。我们的任务是预测值，所以输出层将包含单个神经元。之后我们把该网络的训练算法设置为梯度下降法。我们使用生成的训练数据和标签来训练这个神经网络。接着在训练数据点上运行神经网络，使用 y_pred 接收输出值。程序的第 27～31 行是绘制训练进度并可视化。最后我们绘制实际与预测的输出关系图。

在程序的输出中第一幅图显示了输入数据，第二幅图显示训练进度，第三幅图显示了实际与预测的输出关系。我们可以看到预测输出基本上遵循总体趋势。如果继续训练网络并减小误差，你将会看到预期的输出将更加准确地匹配输入曲线。在终端的输出中显示了 1200 次迭代次数。

BP（Back Propagation）神经网络是 1986 年由 Rumelhart（鲁梅尔哈特）和 McClelland（麦克莱兰德）为首的科学家提出的概念，是一种按照误差逆向传播算法训练的多层前馈神经网络，是目前应用最广泛的神经网络。其基本思想是，学习过程由信号的正向传播与误差的反向传播两个过程组成。正向传播时，输入样本从输入层传入，经隐含层逐层处理后，传向输出层。若输出层的实际输出与期望输出不符，则转向误差的反向传播阶段。误差的反向传播是将输出误差以某种形式通过隐层向输入层逐层反传，并将误差分摊给各层的所有单元，从而获得各层单元的误差信号，此误差信号即作为修正各单元权值的依据。这种信号正向传播与误差反向传播的各层权值调整过程，是周而复始地进行的。权值不断调整的过程，也就是网络的学习训练过程。此过程一直进行到网络输出的误差减少到可以接受的程度，或进行到预先设定的学习次数为止。

提示：

梯度下降法：梯度下降是迭代法的一种,可以用于求解最小二乘问题。梯度下降法的含义是通过当前点的梯度方向寻找到新的迭代点。基本思想可以这样理解：从山上的某一点出发，找一个最陡的坡走一步（也就是找梯度方向），到达一个点之后，再找最陡的坡再走一步，我们不断地这么走，直到走到最“低”点（最小花费函数收敛点）。

18.7　循环神经网络

到目前为止，我们一直在处理的都是静态数据。人工神经网络也很擅长为序列数据建模。尤其是循环神经网络在这方面尤为突出。时间序列数据是我们这个世界上最常见的连续数据形式。当使用时间序列数据时，我们不能只使用通用的学习模型。我们需要描述数据中的时序依赖关系，这样就可以构建一个健壮的模型。

循环神经网络（Recurrent Neuron Network，RNN）是一种对序列数据建模的神经网络（见图 18.13）。RNNs 的目的是用来处理序列数据。在传统的神经网络模型中，是从输入层到隐藏层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能为力。例如，要预测句子的下一个词是什么，一般需要用到前面的词，因为一个句子中前后词并不是独立的。如当前词是“很”，前一个词是“天空”，那么下一个词很大概率是“蓝”。RNNs 之所以称为循环神经网络，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNNs 能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关。

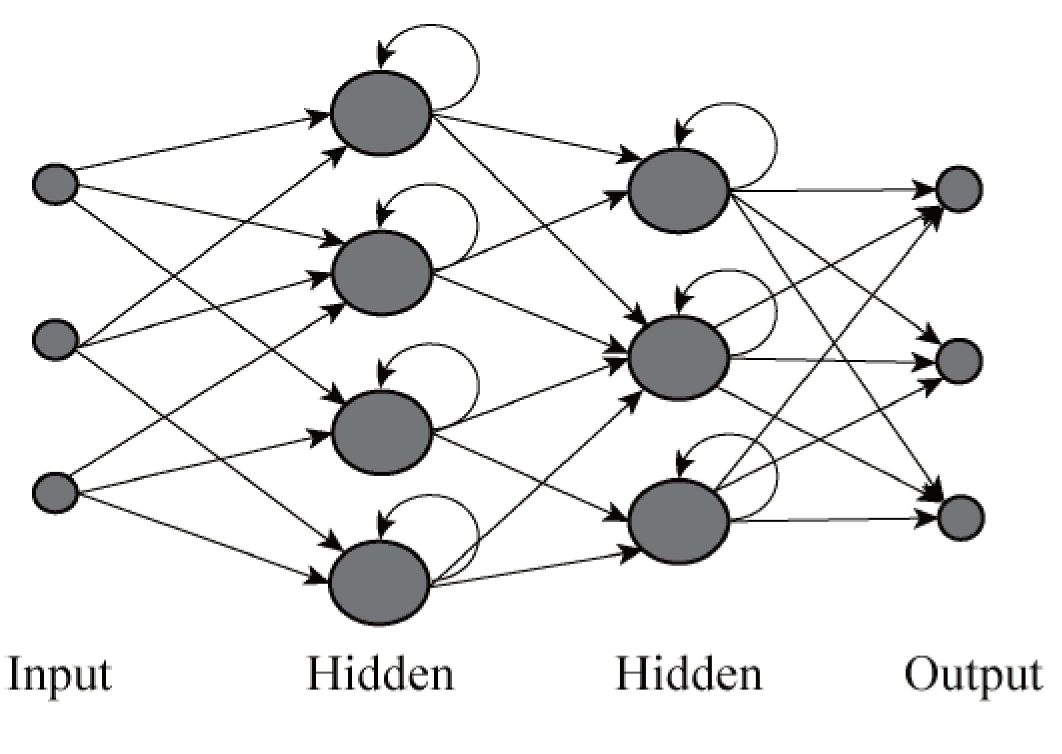


图 18.13　循环神经网络

有别于传统的机器学习模型中隐层单元彼此间是完全对等的，RNNs 中间的隐藏层从

左向右是有时序的，因此隐藏层单元之间是要讲究先来后到的。我们用一段代码来了解循环神经网络是如何工作的，如程序 18.4 所示。

程序 18.4　构建循环神经网络：

```
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

def get_data(num_points):
    wave_1 = 0.49 * np.sin(np.arange(0, num_points))
    wave_2 = 3.62 * np.sin(np.arange(0, num_points))
    wave_3 = 1.2 * np.sin(np.arange(0, num_points))
    wave_4 = 4.6 * np.sin(np.arange(0, num_points))

    amp_1 = np.ones(num_points)
    amp_2 = 2 + np.zeros(num_points)
    amp_3 = 3.1 * np.ones(num_points)
    amp_4 = 0.9 + np.zeros(num_points)

    wave = np.array([wave_1,wave_2,wave_3,wave_4]).reshape(num_points *4, 1)

    amp = np.array([[amp_1, amp_2, amp_3, amp_4]]).reshape(num_points * 4, 1)

    return wave, amp

def visualize_output(nn, num_points_test):
    wave, amp = get_data(num_points_test)
    output = nn.sim(wave)
    plt.plot(amp.reshape(num_points_test * 4))
    plt.plot(output.reshape(num_points_test * 4))

if __name__=='__main__':
    num_points = 50
    wave, amp = get_data(num_points)

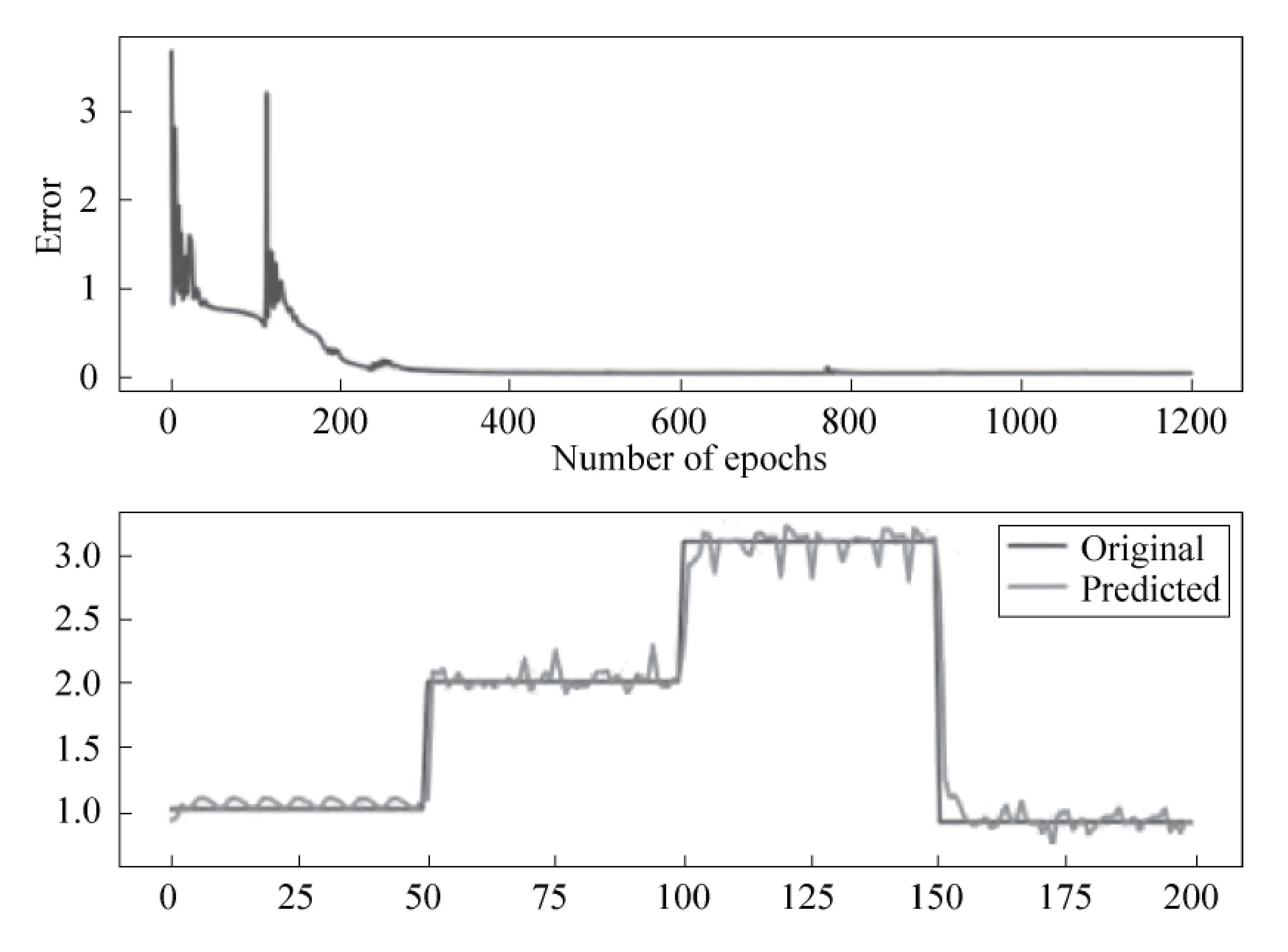
    nn = nl.net.newelm([[-3,3]], [9, 1], [nl.trans.TanSig(), nl.trans.PureLin()])
    nn.layers[0].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
    nn.layers[1].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
    nn.init()
    error_progress = nn.train(wave, amp, epochs=1200, show=100, goal=0.01)
```

```
    output = nn.sim(wave)

    plt.subplot(211)
    plt.plot(error_progress)
    plt.xlabel('Number of epochs')
    plt.ylabel('Error')
    plt.subplot(212)
    plt.plot(amp.reshape(num_points * 4))
    plt.plot(output.reshape(num_points * 4))
    plt.legend(['Original', 'Predicted'])

    plt.figure()
    plt.subplot(211)
    visualize_output(nn, 83)
    plt.xlim([0, 300])
    plt.subplot(212)
    visualize_output(nn, 48)
    plt.xlim([0, 300])
    plt.show()
```

输出，如图 18.14 所示：



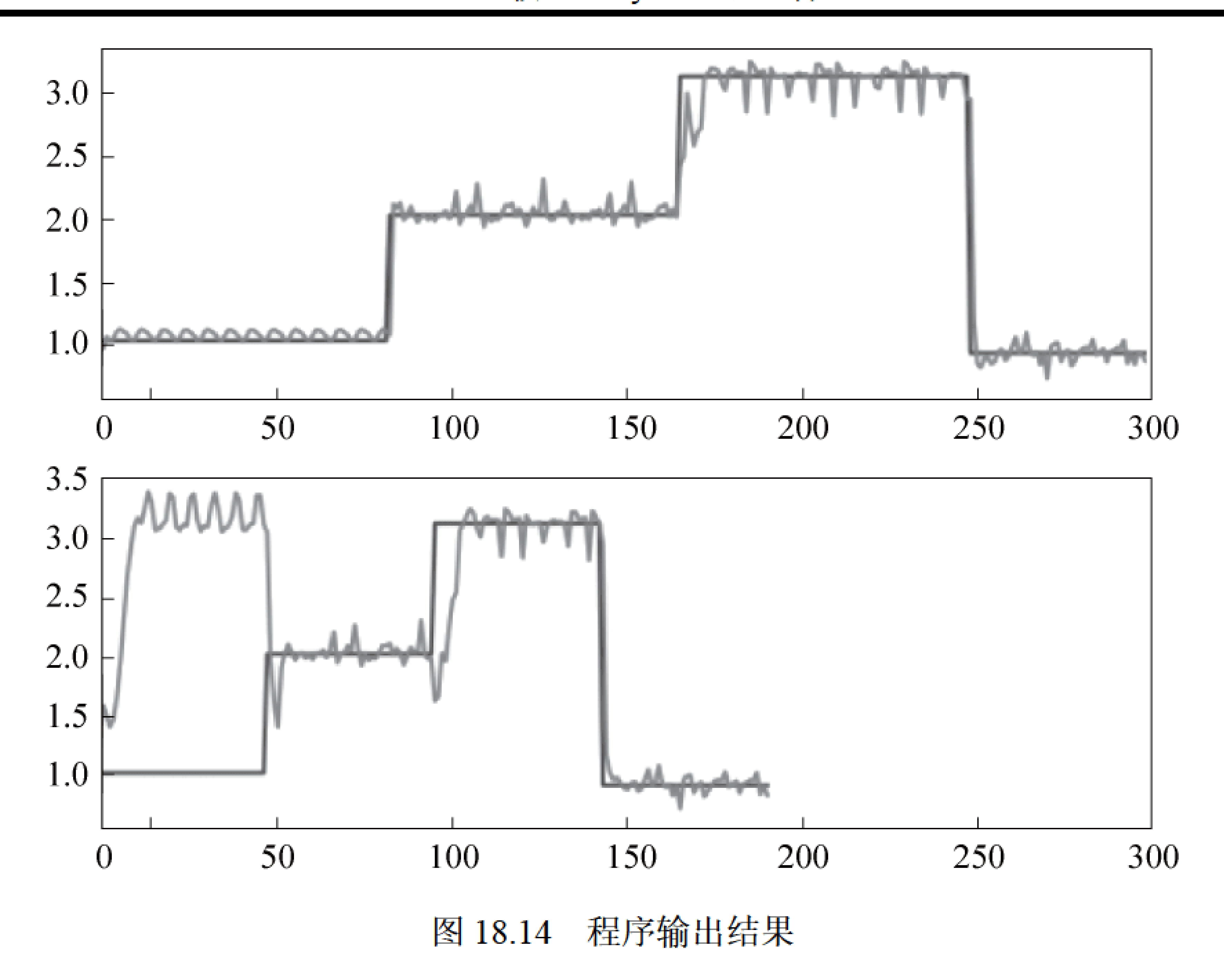


图 18.14　程序输出结果

```
Epoch: 100; Error: 0.6873216412094455;
Epoch: 200; Error: 0.9081901243612933;
Epoch: 300; Error: 0.06353499722527695;
Epoch: 400; Error: 0.05278743365484468;
Epoch: 500; Error: 0.04501848491408758;
Epoch: 600; Error: 0.07467196765059253;
Epoch: 700; Error: 0.03905240909170196;
Epoch: 800; Error: 0.037838042930685434;
Epoch: 900; Error: 0.03913527185304737;
Epoch: 1000; Error: 0.03707538500462401;
Epoch: 1100; Error: 0.035189353608558564;
Epoch: 1200; Error: 0.035847937592551296;
The maximum number of train epochs is reached
```

分析：

该程序是构造一个循环神经网络。首先，导入相关包。然后定义两个函数，第一个函数的作用是生成训练网络所需的训练数据和标签，第二个函数用于可视化神经网络的输出。get_data()函数接收一个 num_points 参数，首先创建 4 个正弦波，再为整个波形创造不同的

振幅，然后生成整个波形和振幅，最后返回这两个值。visualize_output()函数接受两个参数，一个是神经网络，另一个是测试数据点，首先我们得到训练网络所需的参数，然后使用该参数进行网络仿真，接着绘制输出并可视化。上述操作完成之后，我们定义主函数，并定义样本数据点。这里我们定义了 50 个数据点，并使用这些数据点生成了波形和振幅。接着我们创建一个包含两层的循环神经网络。newelm()函数是创建一个 Elman BP 神经网络。为每一层设置初始化函数，程序的第 33～35 行实现了这一点。我们使用之前生成的波形和振幅训练该神经网络，再在网络上运行训练数据，用 output 接收输出结果。程序的第 39～46 行是绘制输出，第 48～55 行是在未知的测试数据上测试神经网络的性能。

在程序的输出中，第一幅图的上半部分显示了训练进度，下半部分显示了在输入波形之上的预测输出。第二幅图的上半部分显示了增加数据点之后神经网络是如何模拟波形的，下半部分显示的是缩短了的波形。在终端上我们可以看到神经网络的训练迭代了 1200 次。

Elman 神经网络是一种带有反馈的两层 BP 网络结构，其反馈连接是从隐藏层的输出到其输入端。这种反馈方式使得 Elman 网络能够探测和识别时变模式。其隐藏层又称为反馈层，神经元的传递函数为 TanSig()，它是一个反正切函数。输出层为线性层，传递函数为 PureLin()，它是一个纯线性函数。这种特殊的两层网络可以任意精度逼近任意函数，唯一的要求是其隐藏层必须具有足够的神经元数目。隐藏层神经元数越多，则逼近复杂函数的精度就越高。Elman 网络和传统的两层神经网络的不同之处在于其第一层具有反馈连接，它可以存储前一次的值，并应用到本次计算中。反馈状态不同，则输出结果不同。因为网络可以存储信息，所以它不但可以存储空间模式，也可以学习时间模式。

18.8　在光学字符识别数据库中可视化字符

人工神经网络可以使用光学字符识别（OCR）。这可能是最常见的例子之一。光学字符识别是识别图像中手写字符的过程。我们可以使用人工神经网络来识别数据库中字符，并将其可视化。在开始构建这个模型之前，需要一个可用的数据集，我们需要从 http://ai.stanford.edu/~btaskar/ocr 中下载一个名为 letter.data 的文件，它是一个光学字符识别数据库。当下载并安装完成所需的数据集，我们就可以开始进行编程了，如程序 18.5 所示。

程序 18.5　可视化光学字符识别数据库中的字符：

```
import os
import sys
import cv2
import numpy as np

input_file = 'letter.data'
img_resize_factor = 16
start = 6
end = -1
height, width = 16, 8

with open(input_file, 'r') as f:
    for line in f.readlines():
        data = np.array([255 * float(x) for x in
                         line.split('\t')[start:end]])

        img = np.reshape(data, (height, width))
        img_scaled = cv2.resize(img, None, fx=img_resize_factor,
                                fy=img_resize_factor)
        cv2.imshow('Image', img_scaled)

        c = cv2.waitKey()
        if c == 27:
            break
```

输出，如图 18.15 所示：

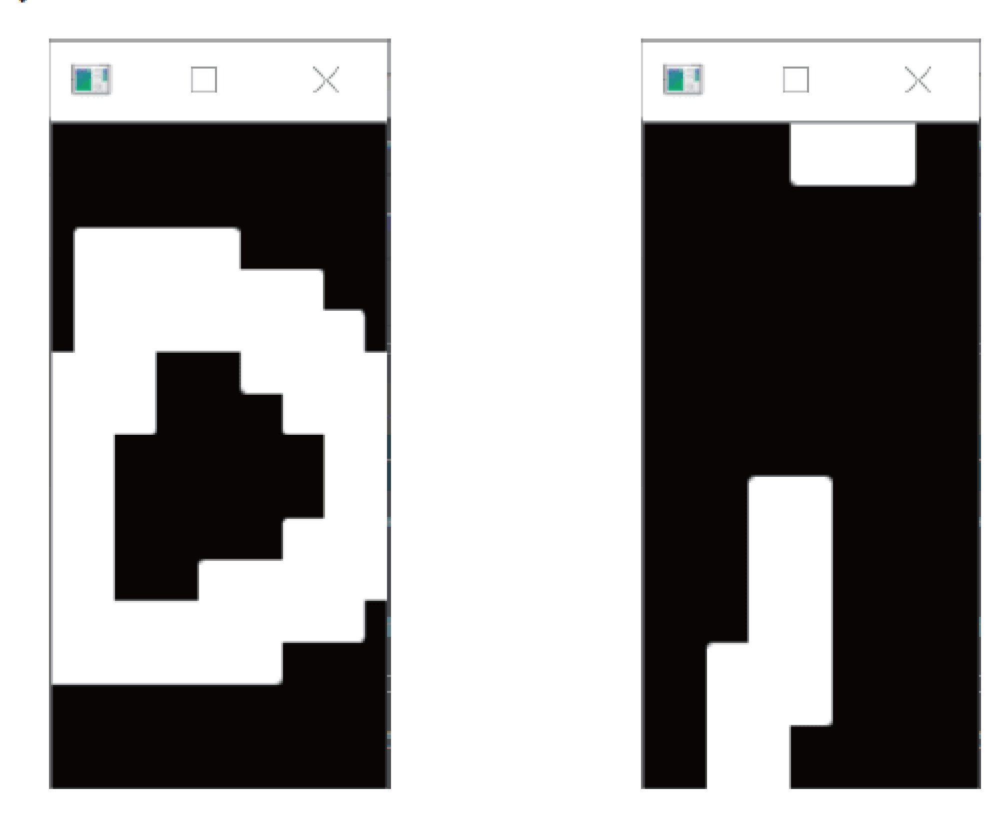


图 18.15　程序输出结果

分析：

这段代码的作用是在光学识别数据库中可视化字符。首先，导入相关的包。其中包括 os 模块、sys 模块、cv2 模块和 numpy 模块。os 模块简单来说它是一个 Python 的系统编程的操作模块，可以处理文件和目录这些我们日常手动需要做的操作，它是对操作系统进行调用的接口。sys 模块提供了一系列有关 Python 运行环境的变量和函数。cv2 是 OpenCV 官方的一个扩展库，里面含有各种有用的函数以及进程。

首先，我们加载光学字符识别数据库中的数据，然后定义一些加载数据所需的其他可视化参数。我们将图像的比例系数设置为 16，从每行的第 6 个元素开始读取数据直到行结尾，图像的高和宽设置为 16：8。接着我们循环迭代该文件中每行的数据，直到用户按下 Esc 键，该文件的每一行都是以 Tab 分隔符隔开的。程序的第 17～20 行是将数组转换为图像，调整其大小并进行可视化。最后检查用户是否按下了 Esc 键，如果是，则退出循环。27 是 Esc 键所对应的 ASCII 码。

运行代码，可以看到一个显示字符的输出窗口。我们可以继续按空格键来查看更多字符。在程序的输出中我们截取了两张截图，第一张是字符 o，第二张是字符 i。

提示：

OpenCV 的全称是 Open Source Computer Vision Library。OpenCV 是一个基于（开源）发行的跨平台计算机视觉库，可以运行在 Linux、Windows 和 Mac OS 操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++类构成，同时提供了 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。

18.9　构建光学字符识别引擎

我们已经学习了如何使用这些数据，现在让我们用人工神经网络构建一个光学字符识别系统，如程序 18.6 所示。

程序 18.6　构建光学字符识别引擎：

```
import numpy as np
import neurolab as nl

input_file = 'letter.data'
```

```
num_datapoints = 40
orig_labels = 'encode'
num_orig_labels = len(orig_labels)

num_train = int(0.9 * num_datapoints)
num_test = num_datapoints - num_train
start = 6
end = -1

data = []
labels = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        list_vals = line.split('\t')
        if list_vals[1] not in orig_labels:
            continue
        label = np.zeros((num_orig_labels, 1))
        label[orig_labels.index(list_vals[1])] = 1
        labels.append(label)
        cur_char = np.array([float(x) for x in list_vals[start:end]])
        data.append(cur_char)
        if len(data) >= num_datapoints:
            break

data = np.asfarray(data)
labels = np.array(labels).reshape(num_datapoints, num_orig_labels)
num_dims = len(data[0])

nn = nl.net.newff([[0, 1] for _ in range(len(data[0]))],
                [128, 16, num_orig_labels])
nn.trainf = nl.train.train_gd
error_progress = nn.train(data[:num_train,:], labels[:num_train,:],
                epochs=5000, show=500, goal=0.01)

print('\nTesting on unknown data:')
predicted_test = nn.sim(data[num_train:, :])
for i in range(num_test):
    print('\nOriginal:', orig_labels[np.argmax(labels[i])])
    print('Predicted:', orig_labels[np.argmax(predicted_test[i])])
```

输出：

```
Epoch: 500; Error: 22.450588249822218;
Epoch: 1000; Error: 4.644646562423083;
Epoch: 1500; Error: 4.609393371036187;
Epoch: 2000; Error: 4.658739623259052;
Epoch: 2500; Error: 4.663467055273863;
Epoch: 3000; Error: 4.660673574025427;
Epoch: 3500; Error: 4.651609857054005;
Epoch: 4000; Error: 4.635088375051949;
Epoch: 4500; Error: 4.595192857240792;
Epoch: 5000; Error: 4.11986688543853;
The maximum number of train epochs is reached

Testing on unknown data:

Original: o
Predicted: o

Original: n
Predicted: n

Original: d
Predicted: d

Original: n
Predicted: n
```

分析：

该程序是用神经网络构造一个光学字符识别器。首先，导入相关包，并加载输入文件。初始化数据点参数，在神经网络处理大量数据时，往往需要花费很多时间来做训练，为了展示如何创建这个系统，我们只用了 40 个数据点。接着我们定义一个包含不同字符的字符串并提取它的长度，也就是不同字符的数量，这个字符串是作为测试数据来测试系统的。接下来定义训练集和测试集的分割，我们将使用数据点的 90%用于训练，10%用于测试。程序的第 11～12 行是定义数据集提取参数，接着创建数据集。程序的第 16～18 行是打开文件，逐行读取，并将每行以 Tab 分隔符分割。如果标签不在标签列表中，我们应该跳过它。程序的第 21～23 行是提取当前标签并将其附加到主列表，程序的第 24～25 行是提取

字符向量并将其附加到主列表。当有足够的数据时，就跳出循环。接着我们将数据和标签转换为数组，并提取数据的维度。程序的第 33～37 行是创建一个前馈神经网络，并将训练算法设置为梯度下降法，使用之前生成的数据和标签训练该网络。最后我们预测测试数据的输出。

在程序的输出中显示了迭代的次数，我们将迭代次数设置为 5000。如果你愿意，也可以设置更大的迭代次数，那么训练结果将会更准确。我们可以看到，字符的预测输出结果是正确的。如果你选择了其他的测试数据，那么测试结果可能会有所不同。

18.10　总　　结

本章我们学习并讨论了人工神经网络、如何构建和训练神经网络、感知器的算法思想以及如何基于此构建一个分类器、单层神经网络和多层神经网络的概念与区别、循环神经网络分析序列数据。最后我们用人工神经网络建立了一个光学字符识别引擎。

18.11　练　　习

（1）自己定义一些训练数据和标签，生成一个感知器网络，并尝试使用不同的参数来训练网络，观察它们的输出（参考程序 18.1）。

（2）简述单层神经网络和多层神经网络的概念和区别。

（3）参考程序 18.6，定义不同的样本测试字符，并增加训练神经网络的迭代次数，观察输出结果有什么不同。

参 考 文 献

[1] 卢克 • 多梅尔. 人工智能[M]. 赛迪研究院专家组，译. 北京：中信出版集团，2015.

[2] 理查德 • 萨斯坎德，丹尼尔 • 萨斯坎德. 人工智能会抢哪些工作[M]. 李莉，译. 杭州：浙江大学出版社，2018.

[3] 周志华.机器学习[M]. 北京：清华大学出版社，2016.

[4] Mahesh Venkitachalam. Python 极客项目编程[M]. 王海鹏，译. 北京：人民邮电出版社，2017.

[5] 张志强，赵越，等. 零基础学 Python [M]. 北京：机械工业出版社，2015.

[6] Wes McKinney. 利用 Python 进行数据分析[M]. 唐学韬，等译. 北京：机械工业出版社，2014.

[7] 林信良. Python 程序设计教程[M]. 北京：清华大学出版社，2017.